Mapping of large task network on manycore architecture

Karl-Eduard Berger

NNT : 2015SACLV026

THESE DE DOCTORAT
DE
L'UNIVERSITE PARIS-SACLAY
PREPAREE A
" L'UNIVERSITE VERSAILLES SAINT-QUENTIN EN YVELINES "

ECOLE DOCTORALE N° 580
Sciences et technologies de l'information et de la communication

Mathématiques et Informatique

Par

M. Karl-Eduard Berger

Placement de graphes de tâches de grande taille sur architectures massivement multicoeur

Thèse présentée et soutenue à CEA Saclay Nano-INNOV, le 8 décembre 2015 :

Composition du Jury :

M., El Baz, Didier	Chargé de recherche CNRS, LAAS	Rapporteur
M, Galea, François	Chercheur, CEA	Encadrant CEA
M, Le Cun, Bertrand	Maitre de conférence PRISM	Co-directeur de thèse
Mme, Munier, Alix	Professeur des universités, UPMC	Examinatrice
M, Nace, Dritan	Professeur des universités HEUDIASYC	Président
M., Pellegrini, François	Professeur des universités, LABRI	Rapporteur
M, Sirdey, Renaud	Directeur de recherche, CEA	Directeur de thèse

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Placement de graphes de tâches de grande taille sur architectures massivement multicoeurs

Mots clés : Placement de tâches, recherche opérationnelle, architectures manycoeurs

Résumé : Ce travail de thèse de doctorat est dédié à l'étude d'un problème de placement de tâches dans le domaine de la compilation d'applications pour des architectures massivement parallèles. Ce problème de placement doit être résolu dans le respect de trois critères: les algorithmes doivent être capables de traiter des applications de tailles variables, ils doivent répondre aux contraintes de capacités des processeurs et prendre en compte la topologie des architectures cibles. Dans cette thèse, les tâches sont organisées en réseaux de communication, modélisés sous forme de graphes. Pour évaluer la qualité des solutions produites par les algorithmes, les placements obtenus sont comparés avec un placement aléatoire. Cette comparaison sert de métrique d'évaluation des placements des différentes méthodes proposées. Afin de résoudre à ce problème, trois algorithmes de placement ont été développés. Task-wise Placement et Subgraph-wise Placement s'appliquent dans des cas où les poids des tâches et des arêtes sont unitaires.
Regret-based Approach est une heuristique de construction progressive basée sur la théorie des jeux qui s'applique sur des graphes dans lesquels les poids des tâches et des arêtes sont variables similairement aux valeurs qu'on peut retrouver dans des cas industriels.
Afin de vérifier la robustesse de l'algorithme, différents types de graphes de tâches de tailles variables ont été générés.

Title : Mapping of large task networks on manycore architecture

Keywords : Tasks mapping, Operational Research, manycore architecture

Abstract: This Ph.D thesis is devoted to the study of the mapping problem related to massively parallel embedded architectures. This problem has to be solved considering three criteria: heuristics should be able to deal with applications with various sizes, they must meet the constraints of capacities of processors and they have to take into account the target architecture topologies. In this thesis, tasks are organized in communication networks, modeled as graphs. In order to determine a way of evaluating the efficiency of the developed heuristics, mappings, obtained by the heuristics, are compared to a random mapping. This comparison is used as an evaluation metric throughout this thesis. The existence of this metric is motivated by the fact that no comparative heuristics can be found in the literature at the time of writing of this thesis. In order to address this problem, three heuristics are proposed. They are able to solve a dataflow process network mapping problem, where a network of communicating tasks is placed into a set of processors with limited resource capacities, while minimizing the overall communication bandwidth between processors. Task-wise Placement and Subgraph-wise Placement are applied on task graphs where weights of tasks and edges are unitary set. Then, in a will to address problems that can be found in industrial cases, application cases are widen to tasks graphs with tasks and edges weights values similar to those that can be found in the industry. A progressive construction heuristic named Regret Based Approach, based on game theory, is proposed In order to check the strength of the algorithm; many types of task graphs with various sizes are generated.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

To Hélène, Stefan and Maximilien Berger,
Ingeborg und Günter Berger,
Mireille and Martial Labouise,
Jeanne Clermon et Huguette Chaumereux,
And my friend Kristina Guseva.

He who loves practice without theory
is like the sailor who boards ship
without a rudder and compass and
never knows where he may cast.

Leonardo de Vinci

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgement

First of all, I want to thank all my Ph.D. advisors. First Renaud Sirdey, my thesis director. He provided me with numerous advices, both professional or personal. His enthusiasm and ideas were always encouraging and he always knew how to reassure, motivate and help me persevere on my thesis when times were hard. Secondly, I would like to thank Bertrand Le Cun, my thesis co-director for his perpetual ideas and suggestions which led to several final results in my thesis. Thanks to his advices, I have learned to adapt to the numerous problems that raised during these three years and remain focused on them.

My foremost appreciation goes to my CEA Ph.D. supervisor François Galea. I was surprised to see how easy it was to work together in spite of our differences. Indeed, during those three years of Ph.D, I have learned more things about computer science than during my master and bachelor years and this, simply by working and chatting with him, observing him and listening to him. All too many times, have I walked out of his office without having understood in depth the meanings of his ideas but, after working on some of my shortcomings, I realized that his ideas were brilliant and would allow me to go forward in my research work. Moreover, and that is unusual: I have deeply appreciated his high degree of commitment and personal investment in my thesis. He always managed to find the time to check on my work and get me some feedback, whether positive or negative. I feel truly blessed to have had him as a Ph.D advisor.

I wish to thank François Pellegrini and Didier El Baz for accepting to review my thesis. A special thanks to François Pellegrini for all remarks and correction he made during his reviews, they have been highly appreciated. I also want to express my sincere appreciation to Alix Munier and Alain Bui for accepting to examine my thesis work. Finally, I wish to thank Dritan Nice for chairing the Ph.D comity.

Now let's talk about the serious stuff. On my first day at the CEA, I was afraid to meet an old bunch of researchers. I was pleasantly surprised to meet people who were actually and dynamic and who had one thing in common: a strong motivation to do research and transmit what they have learned. I immediately got locked in the Ph.D cage that I have managed to escape 2 years later for 6 months before being thrown back inside it again. Therefore I wish to thank all my cage mates: Oana Stan, Safae Dahmani, Simon Fau, Vincent Legout, Dang Phuong N'Guyen, Khanh Do Xuan, Jad Khatib, Paul-Antoine Arras and Rania Mamesh. I also worked for six months in a real office with Laure Abdallah. Having Laure as a co-worker was incredible. The last 10 months in the Ph.D cage were simply awesome thanks to Jad Khatib, Khanh Do Xuan, Rania Mamesh and of course Paul-Antoine Arras.

Another surprise consisted in all the extra trips that have been organized with my colleagues at the time: the amusement parks (i.e. Europa Park, Porto Aventura or Parc Asterix by night) the Christmas Markets of Strasbourg, the long week-end in a huge villa in Ramatuelle and the week-end in Barcelona. Many thanks to all members of these trips: Loic Cudennec, Pascal Aubry, Sergiu Carpov, Cyril Faure, Oana Stan, Julien Hervé, Kods Trabelsi, Selma Azaiez, Safae Dahmani, Vincent Legout, Alexandre Aminot and Aziz Dziri.

I also have to thanks Marie-Bénédicte Jacques and especially Patrick Ruf for all the advices they provided me in improving my writing skills and the quality of my work. And for the long chat we had in their office.

I have to mention my Wood Work Out (WWO) and running mates: Daniela Cancila, Luca Incarbone, Vincent Leboeuf, Alain Giraud, Alexandre Aminot, Aziz Dziri, Andrea Castagnatti and Pascal Aubry.

One special thanks goes to Oana Stan. Indeed, her Ph.D work inspired me during these 3 years and without her advices and her work, my work would have been long and far much more difficult. She fortunately provided me her precious help.

Additionally, I am grateful to Cyril Faure whose advice enabled me to both chair the Ph.D association and to teach during my research work. It really took a lot of effort but with some organization skills, I managed not to burn out.

Another special thanks goes to Fabrice de Mayran de Chamisso. We shared the same ideas and feelings about how things are going and it was always fun to chat and to deepen ideas and concepts. Moreover, he often gave me some tips which have been useful in many of my computations.

Many thanks to Sylvie Esterlin-Thiollier. Working with her for preparing the Ph.D's arrivals and some formations for Ph.D was appreciated and her help very valuable. Our lunches were always interesting, I learned many valuable things and I met several interesting people thanks to her !

The last special thanks goes to Thierry Goubier, who virtually demolished many times my car and often proudly made many pertinent and rightful demonstrations about why I should stop buying German cars and how they suck. Actually, he often had a point.

I wish to cite everyone I met at CEA but there are too many of them. So I apologize for not mentioning everyone's name, but I wish to thanks all the members of the DACLE department and INSTN I met. The coffee breaks, the discussions, the after-works, the picnics, lunches, more coffee breaks, more discussions, more jokes... Thanks for that.

During one and half year, I chaired the CEA Ph.D. association of Ile-de-France. I created a project which gathered many Ph.D associations in order to perform social events (like afterworks or actions for cancer cure). By chatting with many other Ph.D. candidates that I met through the association, I realized what a thesis is and how difficult it is to pursue and to finish it. Moreover, this networking boost allowed me to meet a lot of interesting profiles. I wish to thank all Ph.Ds members of any Ph.D. association.

What would I have done without my Rock'n Roll friends and my favorite trip and dance partner: Cédric Salmon? Without forgetting to mention Emmanuel Gaudry, Adrien Nahlovksy, Gilbert Bienaimé, Rittchy de Triolet, Olivier Bernard and Juliette Bour. A special thanks goes to my roommate Juliette Ledouce for enduring me the last

two years of my Ph.D. I also wish to cite very valuable friends that I met during the time I worked my thesis like Emmanuelle Cazayus, Marie André-Ratsimbazafy, Melisande de Lassence, Emmanuelle Brun, Marc Rivault, Cécile Bouton and Julie Loffi.

I also wish to thank Sophie Grousset, Bénédicte Leclec'h, Ejona Kishta, Anne Vajou, Margot Didier and Hayfa Alaya who provided me with the help I needed in dire times.

A great thank you to Mathilde Adorno for the domino illustration she made foor my paper.

A thank you goes to Julie Loffi who helped me in my thesis defense preparation by inviting me in Japan and for having, with Aziz Dziri, prepared the last details of it. And for enduring me while I was getting nervous a couple of hours before the defense.

Another thank you goes to Hélène and Stefan Berger, Soukaina Bel-Hadj and Julie Loffi which were my syntax error trackers for the last version of this manuscript.

I wish also to thank more long-term friends for having supported and encouraged me throughout all these years: Rafael Perez, Soukaina Bel-Hadj Soulami, Amine Amokrane, Fodhil Babaali, Amaury Vannier-Moreau, Laurent Bourasseau (who kicked my ass many times during my Master time), Nadjet Beghoul and of course the Valdès-Forain Family and the Dargouge Family.

Before ending my acknowledgment, I would like to make a special dedication to one of my dearest friend: Kristina Guseva. We met under very strange circumstances and we quickly became very good friends. She provided me with very pertinent advice and has always been there when I needed someone. She always made fun of me in a way that always made me smile. Unfortunately, Kristina passed away too early at the age of 27 after a difficult battle against cancer. Kristina, I miss you.

Last but not least, I have to thank my family for everything. First to my parents Hélène and Stefan Berger that I truly love. My first student years were chaotic but they helped me to stay focused on studies and thanks to their help, I was able to write this thesis. One never says enough to his parents how much one loves them and is grateful for what they did. To mein little Bruder Maximilien Berger. To my French grand-parents: Mireille and Martial Labouise, my grand-aunts Jeanne Clermon and Huguette Chaumereux. Also to my German grand-parents: Ingeborg and Gunter Berger. I deeply regret that I could'nt finish my Ph.D before the death of meine Oma. Last, I also wish to thank Ina, Maria, Adrien and Axel Berger and other Berger relatives for everything.

The last thanks I wish to provide concern a living creature who just doesn't give a damn about the fact I did a Ph.D or anything else in my life and has a very special way of loving me by ignoring me: my cat Isis.

Publications

- K.E. Berger et F.Galea. “An efficient parallelization strategy of Dynamic programming on GPU”
  - PCO’13,IPDPSW’13,Boston,2013
- K-E. Berger , B. Le Cun, F. Galea and R. Sirdey. “Placement de graphes de flots de données de grande taille”
  - Roadef’14, Bordeaux,2014
- K-E. Berger , B. Le Cun, F. Galea and R. Sirdey. “ Fast Generation Of Large Task Network Mappings”
  - PCO’14,IPDPSW’14, Phoenix,2014
  - Electronique et RO , GT OSI, UPMC, Paris, 2014
- K-E. Berger , B. Le Cun, F. Galea and R. Sirdey. “Un modèle de regret pour le placement de graphe de tâches de grande taille”
  - Roadef’15, Marseille, 2015
- K-E. Berger , B. Le Cun, F. Galea and R. Sirdey. “ Mapping of large task graphs with a regret model ”
  - Soumis à Computer & OR, Elsevier

Résumé

Ce travail de thèse de doctorat est dédié à l'étude de problèmes de placement de tâches dans le domaine de la compilation d'applications pour des architectures massivement parallèles. Ce problème vient en réponse à certains besoins industriels tels que l'économie d'énergie et la demande de performances pour les applications de type flots de données synchrones. Ce problème de placement doit être résolu dans le respect de trois critères : les algorithmes doivent être capables de traiter des applications de tailles variables, ils doivent répondre aux contraintes de capacités des processeurs et prendre en compte la topologie des architectures cibles. Dans cette thèse, les tâches sont organisées en réseaux de communication, modélisés sous forme de graphes.

Pour évaluer la qualité des solutions produites par les algorithmes, les placements obtenus sont comparés avec un placement aléatoire. Cette comparaison sert de métrique d'évaluation des placements des différentes méthodes proposées. La création de cette métrique est due au fait de l'absence d'heuristiques dans la littérature avec lesquelles nous pouvons nous comparer au moment de la rédaction de ce manuscrit.

Afin de résoudre ce problème, deux algorithmes de placement de réseaux de tâches de grande taille sur des architectures clusterisées de processeurs de type many-coeurs ont été développés. Ils s'appliquent dans des cas où les poids des tâches et des arêtes sont unitaires. Le premier algorithme, nommé Task-wise Placement, place les tâches une par une en se servant d'une notion d'affinité entre les tâches. Le second, intitulé Subgraph-wise Placement, rassemble les tâches en groupes puis place les groupes de tâches sur les processeurs en se servant d'une relation d'affinité entre les groupes et les tâches déjà affectées. Ces algorithmes ont été testés sur des graphes, représentant des applications possédant des topologies de types grilles ou de réseaux de portes logiques. Les résultats des placements sont comparés avec un algorithme de placement, présent dans la littérature, qui place des graphes de tailles modérées et ce à l'aide de la métrique définie précédemment.

Les cas d'application des algorithmes de placement sont ensuite orientés vers des graphes dans lesquels les poids des tâches et des arêtes sont variables, similairement aux valeurs qu'on peut retrouver dans des cas industriels. Une heuristique de construction progressive basée sur la théorie des jeux a été développée. Cet algorithme, nommé Regret Based Approach, place les tâches une par une. Le coût de placement de chaque tâche en fonction des autres tâches déjà placées est calculé. La phase de sélection de la tâche se base sur une notion de regret présente dans la théorie des jeux. La tâche qu'on regrettera le plus de ne pas avoir placée est déterminée et placée en priorité. Afin de

vérifier la robustesse de l'algorithme, différents types de graphes de tâches (grilles, logic gate networks, series-parallèles, aléatoires, matrices creuses) de tailles variables ont été générés. Les poids des tâches et des arêtes ont été générés aléatoirement en utilisant une loi bimodale paramétrée de manière à obtenir des valeurs similaires à celles des applications industrielles. Les résultats de l'algorithme ont également été comparés avec l'algorithme Task-Wise Placement, qui a été spécialement adapté pour les valeurs non unitaires. Les résultats sont également évalués en utilisant la métrique de placement aléatoire.

Abstract

This Ph.D thesis is devoted to the study of the mapping problem related to massively parallel embedded architectures. This problem arises from industrial needs like energy savings, performance demands for synchronous dataflow applications. This problem has to be solved considering three criteria: heuristics should be able to deal with applications with various sizes, they must meet the constraints of capacities of processors and they have to take into account the target architecture topologies. In this thesis, tasks are organized in communication networks, modeled as graphs.

In order to determine a way of evaluating the efficiency of the developed heuristics, mappings, obtained by the heuristics, are compared to a random mapping. This comparison is used as an evaluation metric throughout this thesis. The existence of this metric is motivated by the fact that no comparative heuristics can be found in the literature at the time of writing of this thesis.

In order to address this problem, two heuristics are proposed. They are able to solve a dataflow process network mapping problem, where a network of communicating tasks is placed into a set of processors with limited resource capacities, while minimizing the overall communication bandwidth between processors. They are applied to task graphs where weights of tasks and edges are unitary set. The first heuristic, denoted as Task-wise Placement, places tasks one after another using a notion of task affinities. The second algorithm, named Subgraph-wise Placement, gathers tasks in small groups then place the different groups on processors using a notion of affinities between groups and processors. These algorithms are tested on tasks graphs with grid or logic gates network topologies. Obtained results are then compared to an algorithm present in the literature. This algorithm maps task graphs with moderated size on massively parallel architectures. In addition, the random based mapping metric is used in order to evaluate results of both heuristics.

Then, in a will to address problems that can be found in industrial cases, application cases are widen to tasks graphs with tasks and edges weights values similar to those that can be found in the industry. A progressive construction heuristic named Regret Based Approach, based on game theory, is proposed. This heuristic maps tasks one after another. The costs of mapping tasks according to already mapped tasks are computed. The process of task selection is based on a notion of regret, present in game theory. The task with the highest value of regret for not placing it, is pointed out and is placed in priority. In order to check the strength of the algorithm, many types of task graphs (grids, logic gates networks, series-parallel, random, sparse matrices) with various size are generated.

Tasks and edges weights are randomly chosen using a bimodal law parameterized in order to have similar values than industrial applications. Obtained results are compared to the Task Wise placement, especially adapted for non-unitary values. Moreover, results are evaluated using the metric defined above.

Résumé Français

Contexte et modélisation

Il existe actuellement un grand nombre d'applications parallèles développées pour répondre aux problèmatiques des domaines à la fois du traitement de signal et du multimédia. Ces applications ont différentes propriétés et nous n'en citerons que deux: d'une part, les niveaux de parallélisme mis en place pour exploiter au mieux les architectures sur lesquelles elles doivent s'executer et d'autre part l'espace mémoire nécéssaire à leur exécution.

Les travaux qui ont mené à cette thèse portent sur l'exécution de ces applications sur un type d'architecture bien spécifique: les architectures embarquées massivement multicœurs. Ce type d'architectures peuvent se définir comme un ensemble de processeurs pour chacun desquels est associé un cache. Les processeurs sont connectés à une mémoire partagée. Cet ensemble formant un nœud de calcul. L'architecture est donc composée de plusieurs nœuds de calcul reliés soit sous forme de réseau tore, soit sous forme de grille. Plusieurs autres topologies existent, mais cette thèse ne traitera que de l'architecture en tore. On peut citer, comme exemple de cette dernière, la puce MPPA de Kalray qui contient 16 cœurs. Chacun de ces cœurs regroupent 16 processeurs et sont regroupés entre eux par un réseau tore.

Ce type d'architecture massivement manycœurs est spécifique à la programmation parallèle. Leur exploitation est difficile, malgré toutes les potentialités qu'elles offrent. En effet, il est nécessaire d'optimiser la communication inter-nœuds, de garantir l'absence d'inter-blocages et de gérer efficacement les accès concurrents pour en garantir une meilleure utilisation. En outre, l'aspect embarqué de ces architectures rajoute de la complexité à leur pratique : les ressources sont limitées par la mémoire et la bande passante de l'architecture cible.

Une solution proposée par le CEA-LIST est l'élaboration d'un langage de programmation ΣC avec une chaîne de compilation associée. Son modèle est celui de la programmation dataflow cyclo-statique. Dans la programmation data-flow, une application est composée de plusieurs ensembles de tâches parallèles communiquant entre elles par des canaux de communication de données de type FIFO. La synchronisation est effectuée par les données. Dans la programmation dataflow cyclo-statique, le graphe de tâches est défini au moment de la compilation de l'application et demeure inchangé pendant son exécution. La quantité de données produites et consommées est connue et chaque tâche peut, de manière cyclique, s'exécuter avec suffisamment de données sur ses ports

d'entrées et produire des données sur ses ports de sorties.

Dans la chaîne de compilation associée se retrouvent quatre grandes parties : la génération du code, l'instanciation du parallélisme, l'allocation des ressources et la génération de l'exécutable. L'attention de ces travaux se porte sur l'allocation des ressources et plus précisément sur le partitionnement et le placement du graphe de tâches sur l'architecture cible. Les applications actuelles présentent des niveaux de parallélisme de plus en plus importants. Elles sont modélisées par des graphes de tâches statiques de grande taille, sont exécutées sur des architectures massivement parallèles et elles doivent tenir compte des contraintes de capacité des processeurs. Quant au problème du placement, au moment de l'écriture de cette thèse, l'heuristique de placement utilisée dans ΣC ne parvient pas à trouver une solution dans un temps acceptable lorsque le nombre de tâches dépasse 2000. L'objectif de cette thèse est de développer une heuristique capable de surmonter cette limite, de placer les graphes sur des architectures de taille supérieure à 256 nœuds, tout en respectant les contraintes de capacités.

Le problème du placement associé se modélise de la manière suivante :

$$\begin{cases} \text{Min } \displaystyle\sum_{t\in T}\sum_{t'\neq t}\sum_{n\in N}\sum_{n\neq n'} x_{tn}x_{t'n'}q_{tt'}d_{nn'}, \\ \text{t. q.} \\ \displaystyle\sum_{n\in N} x_{tn} = 1 \ \forall t \in T, & (1) \\ \displaystyle\sum_{t\in T} w_{tr}x_{tn} \leq C_r \ \forall n \in N, r \in R, & (2) \\ x_{tn} \in \{0,1\} \ \forall t \in T, n \in N. \end{cases}$$

avec comme paramètres C_{nr} la capacité d'un nœud $n \in N$ pour une ressource $r \in R$, w_{tr} la quantité de ressource r requis par une tâche $t \in T$. Comme variable de décision, $x_{tn} = 1$ si la tâche t est placée sur le nœud n.

Etat de l'art

L'état de l'art peut se résumer part le tableau 1. Les solveurs parallèles SCOTCH et Métis sont capables de placer des graphes de tâches allant jusqu'à un milliard de sommets. Cependant, seul SCOTCH tient compte de la topologie cible, et aucun des deux n'est capable de résoudre le problème du placement en respectant les contraintes de capacité.

D'autres approches développées par le CEA-List pour la chaîne de compilation ΣC prennent en compte les contraintes de capacité et la topologie de l'architecture cible mais ne sont pas capables de passer à l'échelle au delà de 2 000 tâches.

Cette étude montre qu'il existe un besoin de méthodes de placement qui prennent en compte la topologie et les contraintes de capacité de l'architecture cible et qui sont capables de placer des graphes de tâches dont le nombre varie entre quelques centaines et quelques milliers de sommets.

Nom	Contraintes	# de tâches	Topologie
(PT-)SCOTCH - Pellegrini *et al.*	Load Balancing	$> 10^9$	Yes
(Par)Metis - Karypis, Kumar *et al.*	Load Balancing	$> 10^9$	No
ΣC toolchain Partitioning & Placing (GRASP + SA) Sirdey *et al.*	Capacité	< 2000	Yes
ΣC toolchain Parallel Simulated Annealing Galea *et al.*	Capacité	< 2000	Yes

Table 1: Etat de l'art résumé. Les données en vert correspondent aux caractérisques nécessaires pour la résolution du problème

Les contributions de la thèse

Les contributions de la thèse se résument en trois aspects: la mise en place d'une métrique d'évaluation du placement des graphes de tâches de grande taille, des méthodes de placement de graphes de tâches unitaires (les poids des tâches et des arêtes sont fixés à 1) de grande taille et des méthodes de placement de graphes de tâches non-unitaires de grande taille.

Une métrique d'évaluation de placement

En recherche opérationnelle, afin d'évaluer la qualité d'une solution obtenue à l'aide d'une heuristique, on compare généralement cette solution à l'optimal ou à une borne inférieure (s'il s'agit d'un problème de minimisation). Dans le cadre de cette thèse, la borne de comparaison est la valeur minimale de placement. Toutefois, au vu de la taille importante des instances, cette valeur ne peut être connue et le problème de trouver des bornes de qualité suffisante et supportant l'échelle du nombre de tâches reste ouvert.

Afin de définir une métrique permettant d'évaluer la qualité du placement, on s'est inspiré de l'approche différentielle de Demange et Paschos [41]. Ces auteurs tiennent compte à la fois de l'optimal et de la pire solution pour définir un rapport d'approximation, et tentent de s'éloigner le plus possible de la pire solution. La considération de ces deux extrema permet de normaliser l'évaluation de la qualité du problème de RO associé. Néanmoins, pour les problèmes de grande taille, la définition de ces deux extrema est impossible pour le moment.

Des alternatives ont été considérées, alternatives pouvant être appliquées sur des instances de grande taille et qui permettent de garantir l'objectivité de la mesure de qualité. L'idée a ensuite émergé d'utiliser une statistique sur les placements aléatoires. Les tâches sont placées au hasard sans biais algorithmique et le calcul d'une moyenne sur plusieurs tirages permet d'obtenir une référence stable qui ne représente ni l'optimum ni le pire des cas mais représente en quelque sorte un investissement nul en matière d'intelligence algorithmique.

Cette référence a permis de définir une métrique d'évaluation de la qualité des placements effectués par les heuristiques développées dans ces travaux. Cette évaluation dépend donc de l'importance de la distance de la solution par rapport à la valeur donnée par la statistique obtenue.

Placement de graphe de tâches unitaires de grande taille

Une première approche gloutonne nommée Task-Wise Placement (TWP) a été développée. Cette méthode utilise une notion d'affinité de distance afin de déterminer à la fois la tâche à placer en priorité et le nœud sur lequel cette tâche doit être affectée. Les tâches non encore affectées sont stockées dans un espace d'attente. Ce processus est répété de manière itérative jusqu'à épuisement du stock de tâches. L'affinité de distance mesure l'intérêt de placer la tache t sur un nœud n vis à vis des tâches déjà placées. Lorsqu'un nœud est saturé, une autre méthode de placement est utilisée : les nœuds non saturés dans le voisinage direct du nœud sont sélectionnés. Un parcours en largeur commençant avec la première tâche sélectionnée par l'algorithme permet de déterminer un ordre de sélection des tâches. Ainsi, seules les tâches non affectées sont considérées et un nombre équivalent au nombre de nœuds non saturés est sélectionné. Les tâches sont ensuite placées sur les nœuds leur correspondant.

Une deuxième approche gloutonne intitulée Subgraph-Wise Placement (SWP) est une méthode de placement à deux phases. Un sous-graphe est déterminé à partir du sommet qui a le moins de voisins puis est construit grâce à un algorithme de parcours en largeur. Sa taille dépend de la capacité maximale restante sur les nœuds multipliée par un facteur $\frac{1}{2}$. Un calcul d'affinités entre le sous-graphe et les nœuds contenant des tâches est alors effectué et permet de placer le sous-ensemble de tâches sur le nœud avec lequel il a le plus d'affinités.

Les instances utilisées sont des modélisations de grilles ou de réseaux de portes logiques. Les architectures cibles sur lesquelles les instances doivent être utilisées sont des tores carrés de longueur de lien égale à 1. La taille des instances varie de quelques centaines à plusieurs centaines de milliers de sommets.

Les deux approches sont ensuite comparées à l'approche utilisée dans la chaîne de compilation ΣC. Les approches développées dans la thèse sont capables de placer des grilles sur des architectures assez facilement et avec des temps d'exécution globalement plus rapides que ceux de ΣC. Sur des réseaux de porte logique, TWP est jusqu'à soixante fois plus rapide que ΣC avec une solution de meilleure qualité. En revanche, malgré le fait que SWP ne soit pas capable de fournir un placement de meilleure qualité que ΣC, cette approche est plus rapide que l'algorithme de comparaison à une échelle de 10^i, i variant en fonction de la taille de l'instance.

Placement de graphe de tâches non-unitaires de grande taille

Les heuristiques développées précédemment ne sont pas aussi performantes sur des cas non-unitaires, c'est à dire des instances dont les poids des tâches et des arêtes diffèrent de 1. Une nouvelle approche gloutonne a donc été mise au point. Il s'agit d'un placement à une phase qui adapte une notion de regret inspirée de la théorie des jeux et l'applique lors de l'étape de la sélection de la tâche à placer. Au lieu de calculer les affinités des tâches entre elles, le calcul sera effectué en déterminant le coût du placement de la tâche t par rapport à tous les nœuds disponibles. Ensuite, pour chaque tâche présente dans l'espace d'attente, les coûts sont triés par ordre croissant. On calcule ensuite la différence de coût de placement sur les nœuds pour chaque tâche et de manière itérative en utilisant l'approche de Kilby[104]. Ce calcul de différence correspond au regret. La tâche ayant le regret le plus fort sera ensuite placée prioritairement sur le nœud de coût le plus faible.

Pour vérifier la robustesse de ces travaux, un panel plus complet d'instances a été constitué. Les grilles et les réseaux de portes logiques, ainsi que des instances issues du Matric Market de Walshaw, des graphes aléatoires et des graphes de séries parallèles seront utilisés. Ces derniers sont très proches des graphes de type dataflow mais s'en différencient par la présence de liens entre des nœuds non consécutifs. Pour chaque type d'instance, 30 graphes de poids de tâches distincts sont générés. La moyenne de la valeur du placement sur ces 30 graphes distincts est ensuite calculée afin d'éviter les cas particuliers. Le nombre de nœuds varie de 16 à 1024, et leur capacité totale dans le système est fixée à 105% de la somme totale du poids des tâches. Trois ordres de grandeur sont utilisés pour les instances : 10 000, 200 000 et 1 000 000 de sommets. L'heuristique fournit les meilleurs résultats pour des instances de type séries-parallèles, indépendamment du nombre de sommets. Les capacités de l'heuristique ont ensuite été comparées à une version non-unitaire de TWP avec un ordre de grandeur de 10 000 sommets et un nombre de nœuds variables. Il en résulte que l'approche par le regret produit une qualité de placement jusqu'à 5 fois meilleure que celle de TWP et jusqu'à 10 fois plus rapide.

Pour exploiter la variabilité dans l'espace des solutions possibles, une heuristique semi-gloutonne proposée par Hart et Shogan [78] a été utilisée. Cette méthode a généré plusieurs solutions issues d'une procédure aléatoire. La construction de ces solutions peut s'effectuer en parallèle en modifiant les caractéristiques de l'aléatoire. Ce principe a été appliqué à l'heuristique développée dans ces travaux, et un choix aléatoire est effectué pour sélectionner les tâches à placer lorsqu'il existe plusieurs valeurs de regret égales. 30 solutions sont construites en parallèle, desquels on tire la meilleure valeur de placement ainsi que le pire temps d'exécution. Les résultats de cette approche indiquent que, dans la quasi-totalité des cas, il est possible d'améliorer la qualité de la solution au détriment des temps d'exécution qui demeurent acceptables. Il est aussi apparu qu'il existe une corrélation entre la qualité de la solution et son temps d'exécution, dans le sens où moins l'heuristique met de temps à générer la solution, meilleure celle-ci est. Une table de probabilité cumulée d'obtention de la meilleure solution parmi les 10 premiers

lancements a été construite, table qui montre qu'en fonction du taux de dégradation de qualité accepté diminue aussi le nombre de solution requis.

Conclusion et perspectives

Cette thèse présente plusieurs méthodes de placement adaptées pour les architectures massivement multicœurs, ainsi qu'une métrique d'évaluation de la qualité de placement pour les instances de grande taille.

Il est possible de généraliser les méthodes de placement proposées ici sur des architectures hétérogènes ou, dans un autre contexte, de les paralléliser pour placer des graphes de plus grande taille dans des temps humainement acceptables.

De plus, l'élaboration d'une méthode de raffinement pourrait améliorer la qualité de la production et également permettre d'utiliser l'heuristique semi-gloutonne dans un cadre d'optimisation plus poussée des solutions.

Enfin, la métrique d'évaluation pourrait être généralisée sur d'autres variantes du problème de placement, ou sur des problématiques dont les instances ont le même ordre de grandeur que ceux traités dans les travaux ici présentés.

Introduction

Computers have become more and more powerful through the last decades and this trend is not likely to stop in the near future. Despite of all these advances, researchers are still looking for solutions to combinatorial problems. The word "combinatorial" is often synonym with computational difficulty. Combinatorics corresponds to the study of several approaches able to determine and manipulate many configurations of discrete-state events that constitute a combinatorial object. The configuration depends on the combinatorial problem to solve. Combinatorial algorithms can be defined as techniques for the manipulation of combinatorial objects such as permutations, arrangements or combinations. The goal of such algorithms is to find patterns which provide acceptable solutions depending on multiple constraints.

As an example of combinatorial complexity, we will use Terquem's dominoes problem: dominoes is a Chinese game played with 28 rectangular tiles with a line dividing its face into two square ends. Each end is marked with a number of spots or pips which varies from 0 to 6. Each tile corresponds to one unique double combination of tile numbers. The traditional basic game variant is the blocking game. In this variant, all 28 tiles are shuffled face down. Each player picks up 7 tiles amongst the 28. One player starts the game by placing one tile down. The placed tile will serves as starting domino. If the second player possesses a tile with the same end value, he places this tile at this end. The game ends when one of the players has no more tiles. The most common dominoes game variant consists in the double-n form where $n = 6$ whereas several domino games exist with a maximum tile end value $n \in \{6, 9, 12, 15, 18\}$. The higher the number, the more the game becomes complicated. In 1847, Orly Terquem stated the following problem: for any general value of n, is it possible to take the complete set of dominoes and finish the game without any dominoes left on the table? In order to solve this problem, dominoes are modeled by a graph. Vertices are labeled from 0 to n. Each domino corresponds to an edge. A tile with i pipes on one end and j pipes on the other end will be represented by edge (i, j). Moreover, some of the tiles have the same number of pipes. So all vertices i have a loop (i, i). The number of edges including loops is $\frac{(n+1)(n+2)}{2}$, making the graph complete. In order to solve the problem, a Eulerian cycle must be found. Terquem then stated a new problem which appeared to be combinatorial: how many different cycles exist when $n = 6$? A solution has been proposed by a work of M. Reiss in *Nouvelles Annales Mathématiques*(1971) confirmed by another approach, much simpler, proposed by G. Tarry in 1886: the number of possible configurations is $129,976,320$. However, these approaches do not take into account tiles with the same end values. Adding these

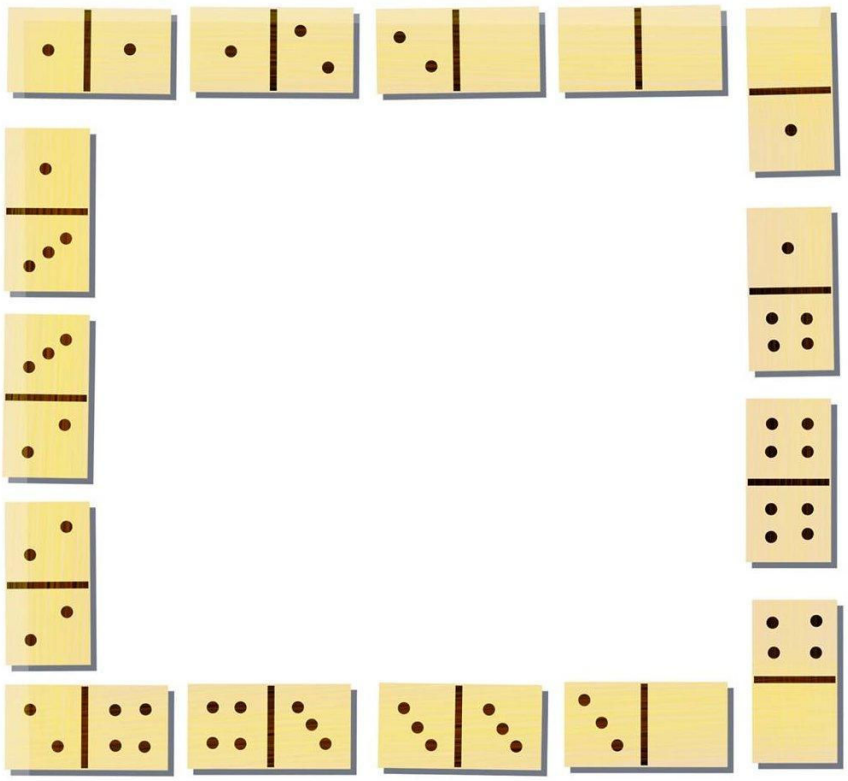


Figure 1: Domino circle with double 4 form.

tiles adds some complexity because it exists 3^7 ways to insert the seven missing doubles into a cycle of 2-combination. This means, it exists $284,258,211,840$ possible ways to use the complete set of 28 dominoes and finish the game with no tile left! Starting from a problem dealing with a small number of elements, it occurs that the possible solution space contains several billions of elements. This example illustrates the fact that combinatorial problems are often associated with large numbers. Now that we described the nature of combinatorics, we will go back to the 2010's.

Nowadays, many applications in the industrial world imply combinatorial subproblems. The resource allocation, the traveling salesman or the knapsack problems are classical combinatorial problems that can be found in many applications. In "the Art of Computer Programming" [108], Knuth proposed a methodology which allows to identify possible solutions to these new combinatorial problems. It consists in five basic types of questions which arise when combinatorial problems are studied:

1. Existence: Is there any arrangement X that conforms to the pattern?

2. Construction: If so, can such an X be found quickly?

3. Enumeration: How many different arrangements X exist?

4. Generation: Can all arrangements $X_1, X_2, ...$ be visited systematically?

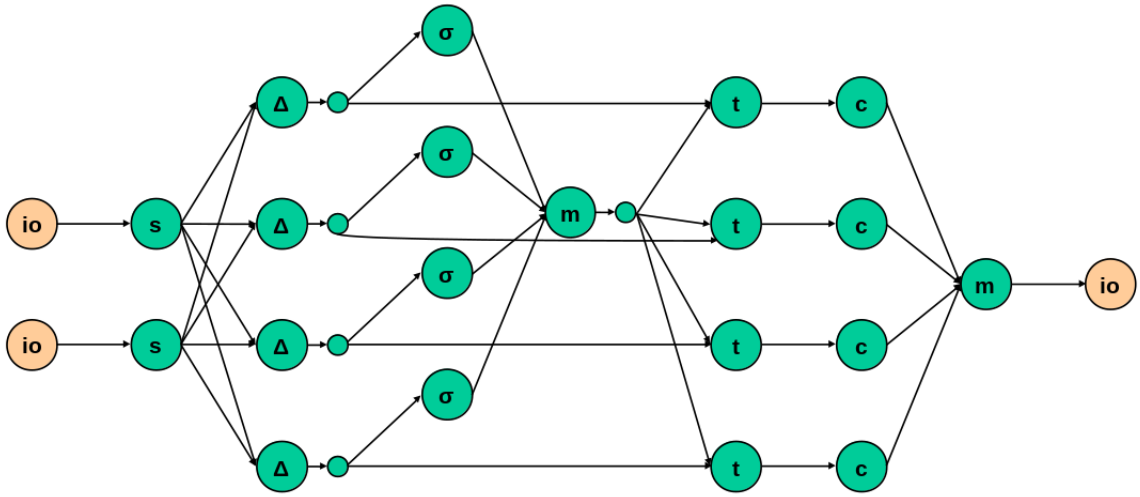


Figure 2: A dataflow process network graph example of a motion detection application.

5. Optimization: Which arrangements maximize or minimize $f(X)$, given an objective function f.

The limits for combinatorial problems are either the required computational time which is not humanly acceptable or overflowing memory size that cannot been dealt by hardware. This is why, despite of the fact that heuristics do not lead to the optimal solution, the obtained values might be the best we can get within those limits. The main question is: how can we obtain an optimal solution within acceptable time or/and without overflowing the memory size for this kind of problems? By the time of writing this dissertation, no one is able to provide an answer.

Limits are often set by hardware. Until the 2000 decade, Moore's Law was a good indicator in the prediction of the performance of new microprocessors. For each new generation of hardware, new optimization challenges also appear for performance enhancement. With the end of the frequency version of Moore's law, new clusterized embedded parallel microprocessor architectures, known as manycores, are currently emerging. New challenges consist in applying combinatorial optimization techniques to problems in software compilation of applications, like in signal processing, image processing or multimedia, on these massively parallel architectures. Such applications can be represented under the static dataflow parallel programming model, in which one expresses computation-intensive applications as networks of concurrent processes (also called agents or actors) interacting through (and only through) unidirectional FIFO channels. They provide strong guarantee of determinism, absence of deadlocks and execution in bounded memory. The main difficulty in the development of this kind of applications for those architectures consists in handling resource limitations, a high exploitation difficulty of massive parallelism and global efficiency. On top of more traditional compilation aspects, compiling a dataflow program in order to achieve a high level of dependability and performance on such complex processor architectures involves solving a number of difficult, large-size discrete optimization problems among which graph partitioning, quadratic assignment

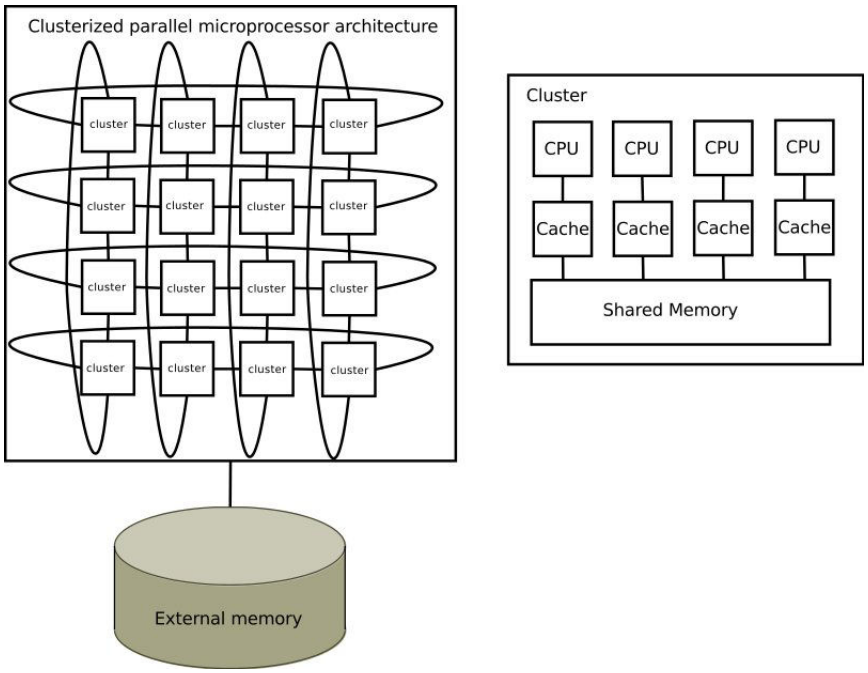


Figure 3: Clusterized parallel microprocessor architecture.

and (constrained) multi-flow problems are worth mentioning.

Our applicative work focuses on the problem of mapping a dataflow process network (DPN), illustrated by Figure 2, on a clusterized parallel microprocessor architecture composed of a number of nodes, each of these node being a small symmetric multiprocessor system (SMP), interconnected by an asynchronous packet network, as shown in Figure 3. A DPN is modeled by a graph where the vertices are the tasks to be placed, and the edges represent communication channels between tasks. Vertices are weighted with one or more quantities which correspond to processor resources consumption and the edges are weighted with an inter-task communication outflow value. The aim of our problem is to maximize inter-task communication inside SMPs while minimizing inter-node communication under capacity constraints to be respected in terms of task resource occupation on the SMPs.

In this dissertation, the mapping is processed under assignment and capacity constraints. A task cannot be placed on several processors and the total amount of all tasks weights must not exceed the capacity of the node. In addition, heuristics able to solve this problem must meet prerequisites. They must be scalable and take into account the target topology. A variety of exact methods, heuristics and parallel heuristics exist to map any type of task graphs on various architectures, but at the time of writing of this thesis, there is no known approach which fits in the constraints and the requirements we settled. For this reason we established the following greedy heuristics: Subgraph-Wise Placement (SWP), Task-Wise Placement (TWP) and Regret Based Approach (RBA) which are the main contributions of this thesis.

These heuristics are based on progressive construction. Subgraph-Wise Placement

is a two phase mapping algorithm. In a first phase, a subgraph is generated using the breadth-first traversal (BFT) algorithm. In order to generate this subgraph, there is a need of a starting task. Size of subgraphs are computed before running BFT. Once the sum of weights of all tasks part of the subgraph has reached this size, BFT is stopped. This task corresponds to the one with the lowest number of neighbors. In the second phase, the generated subgraph is mapped onto the adequate node using an affinity of set property present in the literature.

Task-Wise Placement is a greedy, one-phase mapping algorithm. A starting task with the highest sum of edge weights is determined. It is placed on an arbitrary chosen node. All neighbors are pushed into a waiting set. For each task of the waiting set, a distance affinity value, detailed in this work, is computed. The task with the highest affinity value is selected and mapped onto the adequate node. The two cited algorithms provide results on unitary weighted instances. However, they are not easy to adapt to non-unitary weighted instances. During the task selection step, even if the metric employed improves locally the mapping, no certitude about the global mapping quality can be made. The task selection phase is in the uncertainty. In order to manage this uncertainty, Task-Wize Placement is redesigned by using an approach from game theory. This adaption leads to the design of Regret Based Approach. The initialization phase remains the same. The major differences are in the task selection process. Rather than computing affinity values for tasks in the waiting set, costs values, also defined in this work, are computed. In addition, properties of the regret theory are used in the task selection process. It might happen that mapping a task which seems to be interesting now can lead to a disastrous configuration of the mapping afterwards. In order to mitigate this phenomenon, a regret-based metric is defined. The task with the highest regret value is selected and mapped onto the corresponding processor.

Independently of the algorithms, a new metric for the evaluation of heuristic quality is presented. The contribution of this metric consists in the ability of evaluating mappings quality of large task graphs depending of a random mapping modeled by Erdös-Rényi random graph model.

We compare SWP and TWP to an heuristic denoted as Partitioning And Placing (P&P) present in the literature, developed by CEA-List in previous research. Experiments were made on various task graph topologies generated from the theoretical property graphs (grids, random series parallel), or real world graphs (Logic Gate Networks, or LGN). On grids, our heuristics provide better solution quality than P&P in less time when the number of tasks is greater than 2000. On LGN, TWP provides better solution quality when the number of tasks is higher than $20,000$. Concerning SWP, its run times are several orders of magnitude faster than the P&P approach, while providing solutions whose quality tends to get comparatively similar or better on the largest instances.

The Regret Based Approach heuristic is applied to tasks graphs with random task weights. Indeed, it is currently very difficult to get large sized task graphs from the current dataflow programming development environments, as today's dataflow applications do not show a sufficient level of parallelism yet. Therefore, our methodology was to generate graphs with strong similarities with the topology of typical dataflow programs. In addition, in order to show that our method works better on typical dataflow program

topologies, we compared our results with those obtained using different graph topologies like grids, LGN, series-parallel, random graphs, sparse matrices from the Matrix Market collection. RBA provides the best results on task graphs architectures which shows the most degree of similarity to DPN. Moreover, it provides lower run times and better solution quality than an adapted version of TWP on heterogeneous instances.

This manuscript is organized as follows:

Chapter 1 introduces the context of this work. A brief history of Moore's law and its limitation is made. These limits will lead to the emergence of parallel embedded systems which will be introduced. We will focus on the particular features of massively parallel embedded systems and its associated optimization challenges. The main hardware components of a manycore architecture as well as the difficulties in the development of applications will be detailed. In order to deal with these difficulties, the dataflow programming model will be presented and also the ΣC language designed by CEA-List. This language uses a dedicated compilation toolchain. The mapping process is part of this compilation process. The mapping problem we want to solve is also introduced and the different requirements that heuristics have to fulfill are presented. Another important contribution of this chapter consists in the establishment of a new quality metric based on random mapping. This metric is able to evaluate the quality of a mapping of very large graphs. A reference metric was necessary because at the time of writing of this dissertation, we did not find any comparative algorithm in the literature which met the same requirements than our mapping problem.

Chapter 2 is dedicated to the state of the art of the mapping problem. It starts by detailing the importance of having a good mapping on target architectures. The mapping approach can be divided in three part: the partitioning problem, which consists in splitting the task graph in partitions corresponding to the set of nodes. The assignment problem consists in determining the best way to place subsets of tasks on nodes. Finally the mapping problem which can be split in two types of mapping: first, a two phase approach consisting in partitioning and assigning the task graphs. second, a one-phase approach which directly places tasks one after another. In addition, solvers able to deal with the partitioning and mapping problems can also be found. These solvers are able to provide good solutions and are mostly scalable depending on the size of instances. The focus is laid on some of the most famous among these solvers. A summary in which all elements of the literature, which are cited in this state of the art, is then made. All articles are considered in terms of scalability, topology awareness of the target architecture and constraints that are enforced in it. This summary allows us to locate our work in the global mapping literature.

Chapter 3 presents two scalable capacity constrained mapping heuristic methods able to map unitary weighted tasks graphs on specific target architectures. Before that, a brief review of graph theory elements, which are used in this chapter, is made. Then, the two phase mapping algorithm is presented. Firstly the breadth-first traversal algorithm is used in order to generate a limited-size subgraph. A subgraph to node affinity value is computed and a node to node affinity value too. Both values are compared. If the node to node affinity is lower, than corresponding nodes are merged, otherwise the subgraph is mapped onto the suitable node. The second heuristic is a one-phase process. Each task

is placed one after another. A relation of affinity, which will be explained in the chapter, is used in order to compute the current affinity of each visible unassigned task towards the already mapped tasks. The task with the highest affinity is mapped. Experiments are performed on grids and logic gates network and results are compared to Partitioning And Placing method and the random mapping obtained by the metric defined at the end of chapter 1.

Chapter 4 shows a scalable capacity constrained mapping heuristic able to map non-unitary weighted tasks graphs on specific target architectures. First, the backgrounds of game theory are introduced. The interest is especially laid on behavioral game and decision theory for game theory. Then, regret theory is presented. Different types of regrets are described and the use of this theory in combinatorial problems like scheduling or QAPs are detailed. Once the description is performed, the heuristic is explained. It is strongly inspired by the one-phase heuristic proposed in the previous chapter. Regret values are computed during the task selection process using a notion of costs instead of affinities then, tasks with the highest value of regret are iteratively selected and mapped. Once the RBA heuristic has been established, a Greedy Randomized Adaptive Search Procedure (GRASP) is applied. The aim is to run the algorithm in parallel and to determine how many solution generations are required in order to get a good solution. Experiments are performed on different graph topologies and the metric defined in chapter 1 is used in order to evaluate the performance of the heuristic. In addition to this metric, a comparison is performed with an adapted TWP version which provides comparative results. For the GRASP procedure, we noticed that the number of solutions which have to be generated and the time required to obtain the solutions depends on solution quality, that is to say, it increases with solution quality.

The conclusion summarizes the thesis and presents several perspectives leading to new improvements allowing to handle larger task graphs. It also shows applications of the aforementioned heuristics on other type of architectures.

Chapter 1

Context and Presentation of the Mapping Model

Contents

1.1 Introduction

Nowadays, Embedded Systems (ES) invade increasingly our daily technologies, taking a larger part in our everyday lives. They can be found in critical applications like automotive, aviation, medical life support systems. The environment, in which they are embedded, is constantly evolving, mostly with the emergence of new architectures, new types of processors, that is to say new technologies. As a consequence, ES have to be adapted to these environmental changes and have to be constantly verified because of their critical nature that can be expressed by a loss of properties or damages that may occur to ES on people's lives! Moreover, there is an increasing need for computing power and the guarantee of a high level of reliability. This rapid evolution of the whole system, where ES are integrated, also complicates lifetime of efficient solutions, making them quickly obsolete. This precarious aspect forces both research and industries to improve

and to evolve in order to be able to address these problems. One of the latest proposed solution able to gain performance consists in the massive use of parallelism in embedded systems. This solution will last as long as limits of parallel architecture has not been reached. The last decade have seen the emergence of many tools which facilitates parallel programming in terms of parallel models or programing languages which are able to exploit hardware parallelism.

In this chapter, the context which surrounds this work is first detailed. Then, the dataflow programming model, on which this dissertation is based, is introduced. The intention is to establish the necessary background and terminology for the following chapters. It begins with a brief history of Moore's law and with the emergence of embedded systems. Parallel architectures are also introduced as well as a discussion about programming models that can be used for an efficient exploitation of the characteristics of applications. Then, a parallel programming language and its associated compilation process, developed by CEA-List, based on one of the aforementioned explained programing models is presented. A focus will be set on one step of the compilation process. This step consists in the mapping of the generated tasks graph during compilation towards the target architecture where the application will eventually be executed. Then, the associated dataflow mapping problem, which is the core of this thesis, is detailed. Last, in order to compare the solution quality of all developed heuristics in this thesis, a metric is introduced. The establishment of this metric is inspired by Demange and Paschos, who provided a differential approximation indicating how far the value of a solution is from the worst possible value. However, instead of using the worst possible value, results of the heuristics will be compared to average value furnished by a random mapping. This is the only approach which provides comparable results on unitary and non-unitary weighted task graphs. At the end of this chapter, several elements of graph theory, used for the design of the heuristics developed in this thesis, are defined.

1.2 Moore's law

The remarkable evolution of semiconductor technology from single transistor to multi-billion-transistor microprocessors and memory chips is an amazing story. The invention of the first planar transistor in 1959 led to the development of integrated circuits (IC). 1964 saw the apparition of ICs with 32 components composed of transistors, resistors and capacitors. The year after, ICs contained 64 components. Gordon E. Moore pointed out and provided an interesting definition of this phenomenon. Starting from a simple observation, Moore made an extrapolation, illustrated by Figure 1.1, later denoted as Moore's law, which is still approximatively valid today:

> "*The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the long term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain constant for at least* 10 *years.* "

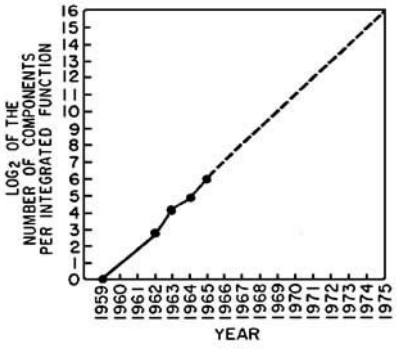


Figure 1.1: Moore's 1965 prediction of the doubling of the number of minimum cost component on chip each year, extrapolated to 1975.

This extrapolation has been made for the Electronics magazine issue of April 1965 [136]. As a matter of fact, ten years later, Intel produced an IC with 65,000 components, making Moore's assumption true.

For half a century, the exponential growth asserted by this law proved to be true. Several studies tried to provide some explanations to justify this curious phenomenon [165], [125]. It is interesting to see how transistors became lighter, faster, able to consume less power and more reliable. Moreover, transistor costs also decreased, making them very attractive to be used in many electronic systems and for various applications.

The first challenging limit for Moore's law comes from miniaturization during the last decade. To put things differently, transistors are expected to be smaller, faster and less power consuming. From the hardware perspective, those characteristics strongly relied on voltage scaling, which is strongly correlated to the size of transistors. However, a physical limit appeared: thermal voltage fluctuations do not scale. In other words, voltage scaling reached this limit and heat dissipation started to physically damage the processor and other devices next to it. The loss of the ability to reduce voltage scaling led to a dilemma: either processor consumption must be reduced or processor frequency must be increased. Actually, none of these alternatives can be achieved anymore and the clock frequency cannot be more cost-effectively increased.

Since the year 2000, the number of transistors in a CPU reached several tens of millions. Nowadays, this order of magnitude reached the billion and is still increasing. Moore's law has evolved. It is not only focusing on the number of transistors per square millimeter, but also on additional features including maximum frequency, power consumption reduction and energy dissipation per mm^2, as shown in figure 1.2. However, while the number of cores increases, making the density of transistor higher, frequency remains the same as well as voltage. Consequently, power density increases for every semiconductor device starting from year 2010. Therefore, a new limit is reached: by having too many transistor on a chip, it is not possible to use all of then simultaneously or the device will either burn or melt. This phenomenon is known as dark silicon [39].

Since the maximum clock frequency has been reached for processors, manufacturers had to find a way to increase processor performance. They designed chips with a higher

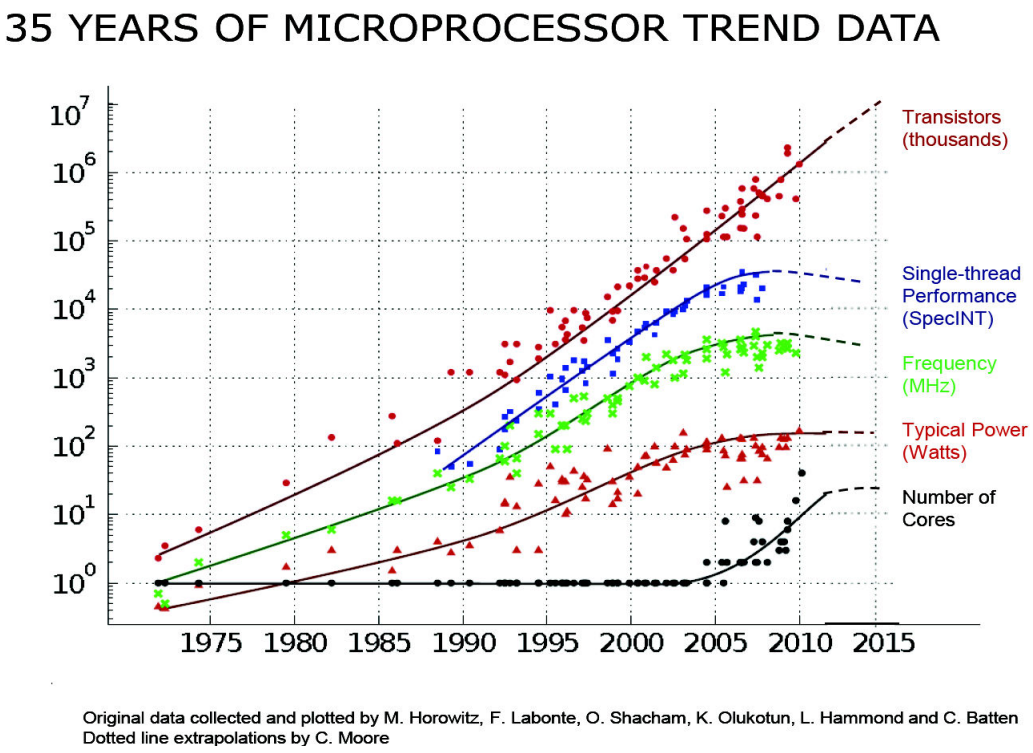


Figure 1.2: Evolution during the last 35 years of the number of transistors, single thread performance, frequency, typical power and number of cores. Image from C. Moore (original data by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten).

number of cores, denoted as multi/many cores, with lower clock frequency.

1.3 Embedded systems

Embedded systems are at the heart of many modern products. Every device that uses a digital interface, like watches or phones, is an embedded system. They can be found in many electronic devices or systems: vehicles, machine tools, cameras, consumer electronics, office appliances, cellphones, GPS navigation, medical equipment, routers, aircraft control systems.

In this section, embedded systems are first defined and then features of parallel embedded systems are presented

1.3.1 What is an embedded system?

An embedded system (ES) is a dedicated computer system composed by a combination of computer circuitry and software. It is a part of a complete device system.

Due to their low cost, ESs are massively produced. They share functionalities with a large range of applications and are often required to provide real-time response in an

order of magnitude of milliseconds or even microseconds. They are designed for several functions like control, monitoring or communication, among other features. In order to compute properly all those features, ESs require powerful general purpose processors. They may be composed of single or multiple processing cores including micro-controllers, field programmable gate arrays (FPGA), application specific integrated circuits (ASIC) or gate arrays. In addition, other components that handle electric interfacing are also integrated.

General-purpose computers aim at managing a large range of processing tasks given by external devices such as keyboards, monitors, hard disk drives. In contrast, embedded systems only perform a small number of well defined tasks. One of the consequences consists in the increase of system robustness. One of the difficulties is that there is a need to minimize costs, power consumption and of course maximizing performance. The complexity of these optimization needs differs, depending on the number of processing chips.

For example, computers have to handle peripheral devices and tasks are much complex to compute. Due to the fact that ESs are designed to perform a tight range of tasks, resource consumption is lower than with general-purpose computers. Embedded systems generally integrate a micro-controller, for reasons of self-sufficiency and cost reduction. Moreover, computing kernels and system tasks are stored in memory without using an operating system (OS). A micro-controller is most of the time composed by slow processors, small memories and interfaces for simple applications which contribute to lower power consumption. This is the reason why the majority of produced microprocessors are used in embedded systems instead of central processing units (CPUs) to control computers.

Despite of the fact that the entire application can be implemented as a single program, some embedded systems include an operating system. The particularity of the integrated OS is that it has been specially developed to be used with embedded systems. They have limited storage, are designed to work in much less memory than desktop operating systems and also work in real-time. They may be able to run an application which contains its own I/O function and do not require a separate OS. VxWorks [166] is the most widely used operating system for embedded systems, developed in the U.S and Europe.

The performance demand of modern complex embedded applications has increased substantially, such that it cannot be satisfied by simply increasing the frequency of a single-core processor. Hence the need for multiple processors that can communicate and provide increased parallelism.

1.3.2 Massively parallel embedded systems

Many embedded systems are designed to run on Systems-on-Chips (SoC). A SoC is an integrated circuit which includes several components like a micro-controller or microprocessor, memory blocks, peripherals, external interfaces, power management, among other devices. All of these components are connected by an on-chip bus.

IC manufacturers want to answer the performance demands of their consumers. Rather than increasing clock frequencies like in the last decades, one way to gain performance

consists in increasing the number of transistors on a chip. Moreover, they started to put multiple and independent processor cores on a single processor in 2005. These improvement ideas led to the creation of multicore processors.

Each core of a multicore processor has a separate flow of control. They access to the same memory. The fact of sharing the same memory adds some difficulty in core exploitation because memory accesses of cores have to be synchronized and coordinated. In order to deal with these new aspects, programmers need parallel libraries, compilers, performance analyzers which help them to efficiently exploit this new multiprocessor architecture. This is the reason why parallel programming methods and tools developed for high-performance computing (HPC) such as multi-threading or OpenMP have been adapted to multiprocessor architectures.

1.4 Manycore architectures

Konrad Zuse and later John Von Neumann developed the first architectures of a computer where the processor is separated from the memory and only executes tasks sequentially [153] [188]. One major drawback of this architecture is that program memory and data memory cannot be accessed at the same time. As the speed of the processor increases, the throughput of the bus between program memory and data memory became smaller than the rate at which the CPU works. This leads to the von Neumann bottleneck, highlighted by John Backus [10].

Programmers have to take into account this bottleneck, however, while new manycore architectures are emerging, the bottleneck issue is reduced. In addition, these new architectures introduce some new abilities to consider. One of these abilities consists in dealing with several instructions and several data in parallel. Before analyzing these architectures, we perform a historical look in order to understand how these architectures appeared.

In the 70's, NASA launched into orbit imaging sensors that generated data at rates up to 10^{13} bits a day. The aim was to extract useful information by using a variety of image processing tasks like geometric correction, image registration, correlation, etc. So there was a need for a tool able to perform between 10^9 and 10^{10} operations per second. In the 80's, NASA Goddard Space Flight Center constructed the first massively parallel computer. This system had to be able to compute thousands of tasks operating simultaneously in order to process image at ultra high-speed by exploiting pixels of the image at the same time. One of the first parallel architecture was born [13], [63] [1].

1.4.1 Parallel architectures

Parallel computers have been used for many years and many different architectures have been proposed. In general, a parallel computer can be characterized as a collection of

[1] In fact, the first parallel architecture was Colossus Mark 2 [35]. This computer, designed during WWII for decryption purposes, was unconventionally modern. It is a single instruction, multiple data (SIMD) machine [196].

processing elements than can communicate and cooperate to solve large problems faster. These computers have several important characteristics to deal with. Some example are the number and complexity of processing elements, the structure of the interconnection network between processing elements, and also the work coordination among them. In order to classify those important characteristics, Flynn proposed a taxonomy [56] which characterizes parallel computers according to the global control and the resulting data and control flows.

Flynn's taxonomy

Flynn's taxonomy splits all characteristics into two fields: Instructions and Data. For each field, a distinction is performed depending whether instructions or data are single or if they are processed in a multiple number of occurrences.

1. **Single Instruction, Single Data (SISD)** There is one processing element which accesses a single program and data storage. At each step, it executes the instruction with its associated data and stores the results back in the data storage unit. The execution is totally sequential and correspond to the von Neumann definition of a sequential computer.

2. **Multiple Instruction, Single Data (MISD)** There are multiple processing elements each of which has a dedicated memory. However, there is only one common access to a single global data memory. Each processing element gets the same data element and loads the instruction from their dedicated memory. Each instruction, which might not be the same, are computed in parallel. Nowadays, it is hard to find any working devices which runs using this architecture.

3. **Single Instruction, Multiple Data (SIMD)** There are several processing elements, each of which has an access to either shared or distributed data memory. In contrast to MISD, the same instruction is synchronously computed in parallel by all processing elements to different data elements. The best example is modern graphic processing units (GPU).

4. **Multiple Instruction, Multiple Data (MIMD)** There are multiple processing elements, each of which has a separate instruction and data access to a either shared or distributed program and data memory. At each step, each processing element loads a separate instruction and data from memory, applies the instruction and stores the result back in data memory. The processing elements work asynchronously. One example is the Intel Xeon Phi processor.

These characteristics are summarized in table 1.1.

Parallel programming systems

Many parallel programming systems exist. The major differences consists in memory policies and processor management. These spectrum of memory and processor structures can be split into three parts as follows:

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	MISD	MIMD

Table 1.1: Summary of Flynn's taxonomy

Shared Memory Machine (SMM) Computation resources are shared physically by all processors [85]. Memory is organized as a central resource for all processors. A node is a large set of processors which materially share the same memory. In shared memory systems, communication between cores is performed using shared variables in memory. Each process reads and writes on the same memory address. It is simpler and faster, but concurrent accesses of different processes can lead to unpredictable results. When the number of processes is high and the same variable is accessed intensively, synchronization operations can slow down the runtime and the risk of congestion increases. In modern architectures, there is a limit in the number of processors which is related to technology. Beyond this limit, no performance gain can be expected. However , Non Uniform Memory Access (NUMA) architectures allows physically distributed processors to be used following the principle of SMM. The maximum number of processors that can be used may be increased without any risks of deterioration of performance.

Distributed Memory Machine (DMM) The processing elements are interconnected by a bus. The memory is physically distributed on processors. A processor is strictly encapsulated and accesses only its own code and local memory. In order to retrieve resources from other processor's local memory, it performs a message passing operation through the bus channel which interconnect all processors [36]. Getting data from its local memory is faster than using the message-passing architecture. One drawback of this architecture is that performance is bounded by the communication channel when resources which are not located in local memory are massively requested.

Hybrid Memory Machine (HMM) It is a system which exploits both types of memories for performance gain. In order to reduce risks of congestion, modern architectures have to deal with a hardware limit on the number of processors. Data flows on communication channel is DMM has to be reduced for performance gain. The idea is to merge these two architectures and to propose a compromise between different drawbacks of these machines. MIMD machines are running with this memory configuration [152] because each processor runs its own program flow. It makes it harder to program but increases flexibility.

1.4.2 Massively Parallel Processor Architectures (MPPA)

A manycore architecture is a hybrid memory machine which can be simply defined as an integrated circuit composed of a massively parallel array of hundreds or thousands processing cores like CPUs and RAM memories interconnected by a static network. A node represents a cluster of processors which corresponds to a SMM. Each processor has

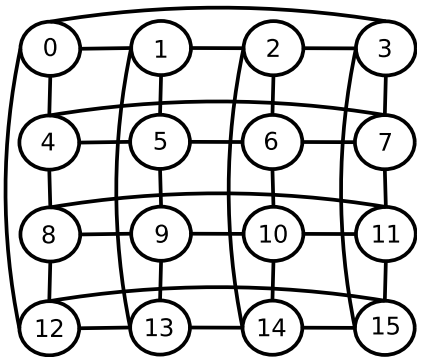


Figure 1.3: Undirected 2-dimensional cyclic mesh. Each vertices correspond to a cluster and each edge to a communication channel.

its own cache and is connected to a cluster-shared memory. The clusters are connected to each other using the DMM logic through an interconnection network. This network corresponds to a network on chip (NoC). In addition to this system, the NoC is also connected to an external memory.

The strengths of such an architecture are based on several concepts. Data manipulations by processors in the same clusters are efficient and fast due to the shared memory features. The number of processors remains small but the size of available memory is higher. For example, the network of clusters can be represented as an undirected 2-dimensional torus as illustrated by Figure 1.3. In this thesis, we choose to select the mesh topology because it is often used in DMM and it facilitates massive parallel computation of matrices.

In this architecture, during the execution of the application, the ideal is to uniformly distribute data across processors. This is why data location has to be carefully chosen by the programmer. For applications which exhibit a large level of data parallelism like image processing, images are decomposed into blocks of pixels. Neighboring pixels are placed in adjacent blocks. These operations are easy to perform. Unfortunately, many real world applications are not as simple as image processing operations. An efficient distribution of data is required in order to optimize the performance of this architecture. Most of the time, determining such an efficient distribution is difficult and, in addition to data mapping techniques, routing techniques also have to be applied. Some advantages of this architecture consist, depending on the application to run, in energy saving, latency reduction or throughput increase. However, when the application is not well programmed, it leads to a misuse of the number of cores, the memory or the bus system, provoking a drastically important increase of the risk of bottlenecks. As a consequence, performance might be deteriorated. One way to avoid this consists in having an efficient routing algorithm able to manage inter-cluster communications. Communication in a NoC is packet-based and routing algorithms are used in order to determine a path in the network from each cluster to the others. The goal of the routing consists in determining the shortest possible route in order to avoid congestions. In the path determination, network contention and network congestions are important and well-known issues.

Nowadays, manycores are used in high-performance embedded systems. As example of massively parallel applications that can exploit this architecture, we can cite network processing, medical imaging, image processing, video compression, streaming media applications among many others can be cited.

The Kalray MPPA-256 manycore is one example of existing massive parallel processor architecture. It is a single chip manycore processor which integrates 16 SMP clusters with 16 cores and 4 I/O subsystems including a SMP quad-core which contains an on-chip memory and a DDR controller and a network on chip. The technology behind this chip has permitted control and flexibility on the voltage. This feature allows to process on the chip, applications with low energy per operations and time predictability requirements.

1.5 SDF and CSDF dataflow programming models

Efficient computation on manycore architectures in an embedded system is a great challenge. Being able to program efficiently in parallel an application which runs on a massively parallel architecture is difficult. Attention must be payed to several features like architecture or resource allocation. The hard part of the work of the programmer consists in synchronizing efficiently all tasks in order to avoid delays or deadlocks. In order to simplify this part, there is a need to find a way to model the application which allows the developer to avoid all problems related to coordinating data. One accurate way to model an application consists in using graph theory. That is to say, any program can be split in several elements modeled by graphs.

Basically, a graph is a representation of a finite set of objects interconnected by links. Objects are denoted as vertices (or tasks) and links as edges. A graph can be employed as an abstract representation of a parallel program. A program can be split into two activities: computation and communication. Computation can be defined as a program which takes data as input and produces data as output. Tasks are usually executed in sequential order. Communication corresponds to the relation and/or data dependence between two tasks and is represented by the edges. The transformation of a program into a graph is difficult and must obey to several set of rules. We will not present all these rules but more details can be found in [134] or [50]. Focus will be set on how program costs in terms of computation and communication are represented. Most of the time, costs in programs refers to time measurement, that is to say, the duration a computation or a communication takes during the execution of a program on specific target architectures. This time cannot be negative and respectively corresponds to vertex and edge weights. Time is not the only cost. Memory occupation, processor occupation, energy cost, bandwidth among other types of resources can also be represented as program costs. In the literature, a large number of graph theoretic models for parallel computation can be found [134], [107] or [90].

Sinnen and Sousa's taxonomy [173] provides an exhaustive recursive classification. It is based on three main criteria: computation type, parallel architecture and algorithm. In this classification, all models can be shared in three classes: dependency graphs, flow graphs and task graphs. Of course, these types can be subdivided into more parts.

In this work, the focus is laid on flow graphs. Among the flow graph, tasks correspond to a sequential computation. The particularity of this type of graph lies in communication. Time information is represented but also communication distance are employed. In flow graph, one model presents features we are interested in: the Data-Driven Execution Model (DDEM).

In DDEM, an edge corresponds to a communication between two nodes via an intermediate queue. Communication data are written by the source node of the edge and are put into the queue from where they are read by the destination node. All data in the queue are processed using a First-in First-out policy (FIFO). A node is enabled to "fire" the execution if each input edge or channel contains data (or `tokens`) and each output edges or channel has one space in the queue. One main characteristics of DDEM consists in the fact that the flow of the data invokes the run of a node without need for synchronization. In addition to that, communication between nodes are performed asynchronously. DDEM provides a view of the data flow in space and time for hardware-oriented parallelization. A derivative of this model turns out to be the Dataflow Process Network.

Dataflow Process Networks (DPN) [113] are mostly used as an execution model of many applications of the multimedia or signal processing family. They correspond to a special case of Kahn Process Networks (KPN) [69]. A KPN is a deterministic general purpose model for parallel programming. Each process only sees a sequence of data values coming in the queue. The process has no visibility on other data which are in the waiting queue. In a KPN, each process consists in repeating "firings" of a dataflow "actor". "Actors" of our DPN correspond to non-trivial computation like filtering, FFT, join-fork operations. Those operations can be represented by tasks to perform and are weighted with one or more quantities which correspond to their resource consumptions. Directed edges are used to model data rates communicated between tasks. When an actor is fired, a certain amount of data tokens are consumed on its input channel and a number of result tokens are produced on the output channel.

This operation induces a form of synchronization. Several nodes can fire at the same time if all of their requirements are met, making the model able to handle concurrency. Many variants of dataflow models exist. Two majors dataflow models are the following: Synchronous Dataflow (SDF) and Cyclo-Static Dataflow (CSDF).

SDF results of a work of Lee and Messerschmitt [114]. It is a restriction of KPN to allow compile time scheduling and is illustrated by figure 1.4. The basic ideas are each process reads and write a predefined number of tokens each time an actor fires. The quantity of produced and consumed data on input and output channels remains constant between two consecutive actor firings. SDF has no initialization phase and starts with tokens which are already allocated in its associated buffers, generating delays which result to avoid bottlenecks.

In the same way, CSDF has been created by Bilsen, Engles, Laiwereins and Peperstraete [17]. It is an extension of SDF dataflow model able to describe applications with a behavior that cyclically changes. For this model, it is assumed that the number of tokens produced and consumed by an actor is known at compile time but c hanges periodically. Figure 1.5 illustrates this behavior. In a more technical aspect, let n be the number of

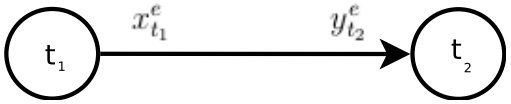


Figure 1.4: Synchronous Dataflow illustration.

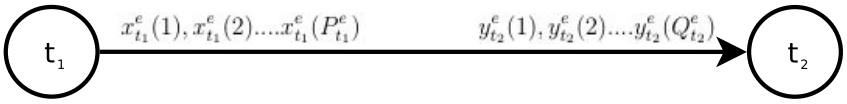


Figure 1.5: Cyclo-Static Dataflow illustration.

times a task t_1 is fired. x and y correspond to a fixed amount of tokens. Let $P^e_{t_1}$ be the number of tokens sent on an edge e by task t_1 and $Q^e_{t_2}$ the number of tokens received on an edge e by a task t_2. For an edge e, a task t_1 produces $x^e_{t_1}(i), i \in [1; P^e_{t_1}]$ tokens every $(n \times P^e_{t_1} + i)^{th}$ time it is called. The consumption behavior of task t_2 is: $y^e_{t_2}(i), i \in [1; Q^e_{t_2}]$ tokens every $(n \times Q^e_{t_2} + i)^{th}$ time it is called.

In the following, the ΣC programming model and language, able to work on dataflow programming model is presented. This language is based on process networks with process behavior specifications.

1.6 ΣC programming language and compilation

The ΣC programming language has been designed by CEA-LIST [71]. This language is based on the CSDF model. The main characteristics of this language consist in a large expression ability able to deal with a huge scope of applications in data processing and multimedia, a component based approach which will provide a way of representing of complex applications, a natural massive parallelism expression, a smart use of resources during compilation and a syntax as close as possible to that of the C language, in order to facilitate programmers work.

The general philosophy consists in exploiting the dataflow parallelism and express it easily so it could be adapted to the target system resources. Figure 1.6 is an example of a ΣC motion detection application.

1.6.1 ΣC programming language

ΣC is a dataflow programming language designed for parallel programming of high performance embedded manycore processors and computing grids. It is a process network-based language which is able to express SDF and CSDF. Moreover, it is adapted for a large spectrum of applications and is able to verify features like the absence of deadlocks or memory bounded executions. It is related to StreamIt [183] and is designed as an extension of the C language. In this extension, tasks in the stream model, which correspond to actors, are denoted as agents. In addition, a set of keywords have to be defined by the programmer, as well as each production and consumption of tasks.

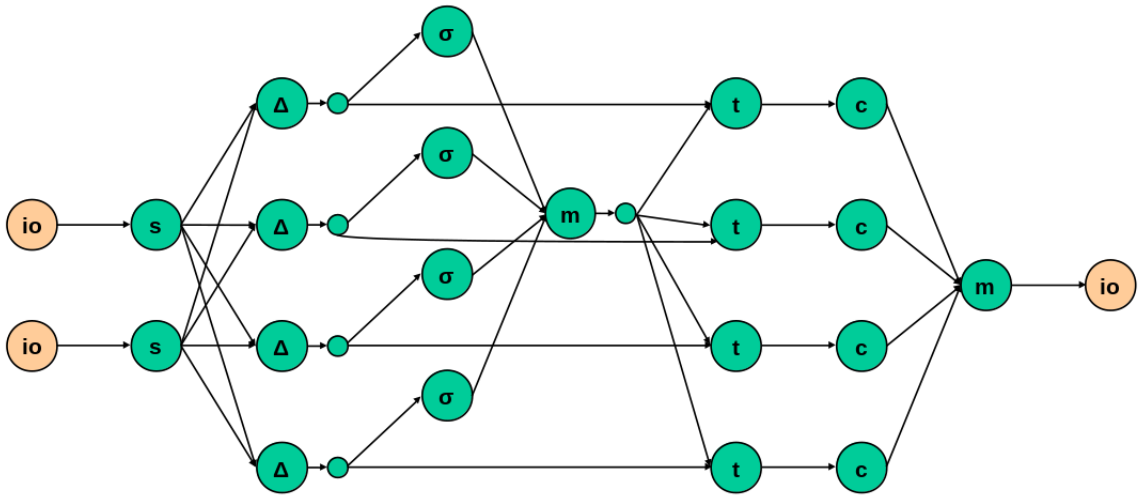


Figure 1.6: A dataflow process network of a motion detection application to map on a torus node architecture target.

An application is described as a static instantiation graph of interconnected tasks. For each cycle, the amount of data which are produced and consumed by tasks remains constant. Tasks are instances of agents and have a cyclic behavior with a variable amount of data due to CSDF properties. For data distribution and synchronization, several dedicated pre-defined agents are provided: *Split*, *Join*, *Dup*, *Select*, *Merge* and *Sink*. More details about this language can be found in [6].

We present in Figure 1.6, an example of application which is the Laplacian computation of an image.

1.6.2 An Example ΣC Application: Laplacian of an Image

The Laplacian is a 2-dimensional isotropic measure of the second spatial derivative if an image. In this image processing operation, input arrays consist in graylevel images. The output array consists in another graylevel image. Laplacian computation highlights zones of rapidly changing intensities of the processed pictures. This feature makes this operation being mostly used for edge detection. The operation consists first in applying an image a Gaussian smoothing filter in order to reduce imperfections. This filter computes a $2D$ convolution operator in order to blur the image. Convolution is one of the image processing basis operation.

The Laplacian operation simply consists in processing two unidimensional convolutions on each row of an input image, resulting in two separate images, which are then processed column-wise using a similar convolution on each image. The two resulting images are summed together and this sum is the Laplacian of the input image. All convolution operations on image lines are independent from each other, as well as the subsequent operations on columns, enabling a high degree of parallelism.

In a more accurate way, the operation first compute the convolution operations on

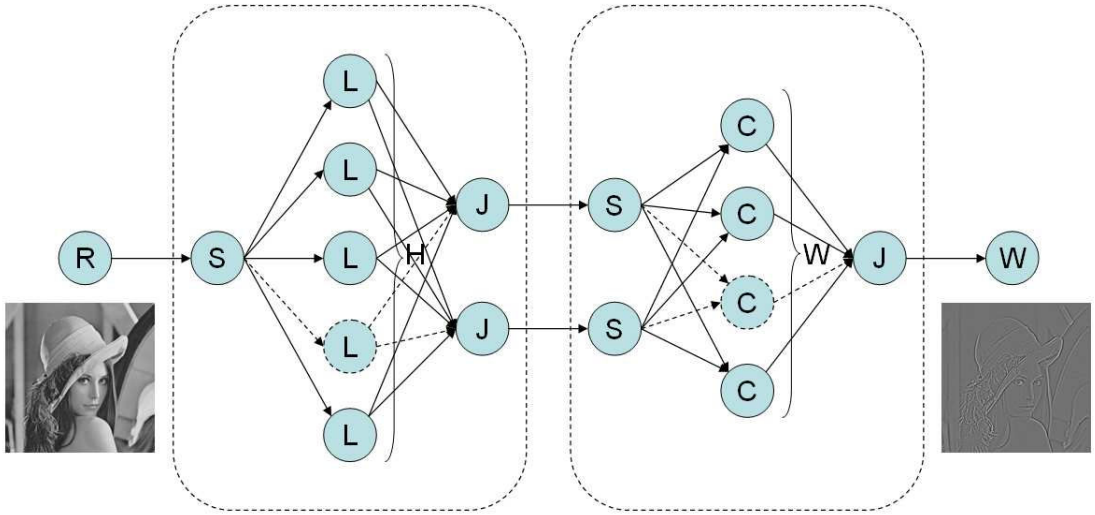


Figure 1.7: Dataflow process network for the Laplacian computation of an image.

lines:

$$L_{i,j}^{(1)} = \sum_{k=1}^{11} I_{i,j+k-1} g_{11-k+1}^{(1)} \tag{1.1}$$

and

$$L_{i,j}^{(2)} = \sum_{k=1}^{11} I_{i,j+k-1} g_{11-k+1}^{(2)}, \tag{1.2}$$

where variable $I_{i,j}$ denote the pixels of the input image, $g^{(1)}$ and $g^{(2)}$ are convolution vectors of size 11. Then, the final image is processed by convolution operations on columns of the two obtained intermediate images:

$$C_{i,j} = \sum_{k=1}^{11} L_{i+k-1,j}^{(1)} g_{11-k+1}^{(2)} + \sum_{k=1}^{11} L_{i+k-1,j}^{(2)} g_{11-k+1}^{(1)} \ . \tag{1.3}$$

Figure 1.7 illustrates the associated dataflow process network of the Laplacian function. The following description of the figure starts from the left side to the right side of the illustration.

Task R represents the task which deals with the input image. Its role consists in reading the input image. Task S, next to R, is a split operation which calls the `Split(W,H)` operator. This function splits the set of data obtained by task R into subsets of data of size W (i.e, the number of lines in the image) and sends each subset on each H output channels of S using a round-robin policy. On the next level, one can observe five L tasks, which can be computed in parallel. The aim of these tasks is to perform the first mono-dimensional convolution phase which applies both masks and generates two images using equations 1.1 and 1.2. This is the reason for the presence of one input channel and two output channels. The next phase consists in the use of 2 `join(W,H)` operations. Subsets of data of size W received by the input channels are inserted, using again a round-robin

policy to reconstruct full images. This leads to the creation of two intermediate images. Next, the `Split(1,W)` operator are applied to both images resulting in the splitting of the pictures into columns. The split operator sends on its W output channels the data to be processed by the C tasks. These tasks will perform the convolution operation and the summation on rows using equation 1.3. Task `join(1,W)` gets the data computed by C and the resulting image is send to task W which will write down the new image.

This example illustrates how the programming language implement dataflow process networks. However, additional features have to be taken into account. For instance, for a 640×480 image provided by a surveillance camera, the Laplacian operation is processed on 640 tasks for the first operation and 480 for the second. Computing these operations in parallel should be possible even if the camera produces around 30 images per second.

After having presented the ΣC language, we now present the compilation toolchain associated with this language.

1.6.3 The compilation process

This section presents a brief overview of the ΣC compilation toolchain. The aim is to link an application written in this language to the execution model, in order to build a binary executable. The following lists all steps necessary to produce the binary executable is built from the execution model.

The first compilation phase of any traditional compiler is a preprocessing phase followed by lexical, syntactic and semantic analysis. ΣC sources are transformed in C codes. Once these codes are generated, it is impossible to alter them unless doing some code substitution. Two types of codes can be listed:

- **Instantiation codes** of the dataflow process network from the compilation of the ΣC keywords and functions which will be mapped off-line onto the target architecture;
- **Processing codes** which come from the agent processing functions compilation.

The next phase processes instantiation codes. Adapted API are called in order to find a transformation into an appropriate C code. Once this code is written, it is compiled into a data structure which represents the process network. It is also compiled in order to obtained data specific to agents. During this compilation phase, parallelism is reduced, for instance by minimizing the number of split's output channels or by merging tasks. This reduction allows to better meet the system resources, to provide access scheme to data processed by `split, join, ...` agents. Another action which occurs during this compilation phase consists in hierarchical consistency checks, verification of the absence of deadlocks and pre-dimensioning measures on the channels associated buffers. At the end of the phase, it is possible to proceed to a first execution.

The third phase is resource assignment. First, communication buffers are dimensioned using tasks temporal characteristics and throughput objectives of the application. It allows the attenuation of the variance effects of the runtime on the average throughput in order to put them below to the critical threshold computed using the constraints of the

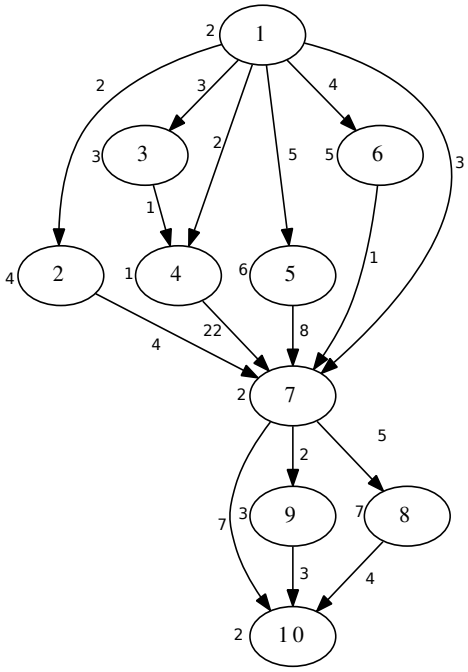


Figure 1.8: A task graph for a fictitious program. Vertices are numbered, vertex and edges weights are noted besides them.

application. Second, in order to link the execution model, a partial unbounded order is constructed on task occurrences [64]. It is processed in order to facilitate the scheduling of tasks on the chip, and to guarantee its execution in bounded memory on the FIFO communication channels. The third phase consists in a mapping and routing phase. The work of this dissertation takes places in this phase. Its aim is to gather, under cluster capacity constraints of the target architecture, tasks which mostly communicate together on the same clusters, while taking into account the distance between pairs of clusters. This step requires solving some difficult discrete optimization problems. Application are represented using a task graph. The vertices represent the tasks of the program and the edges, the communication channels between the tasks. Vertex weights are associated to computational costs and edge weights are associated to communication tasks [172]. Figure 1.8 provides an example of such a weighted task graph. The last phase consists in, on the one hand, the generation of the parameterization data of the system [44] and, on the other hand, building, using a C compiler back-end, a binary executable that can be loaded on the target system [175].

Now that the compilation process has been explained, we will focus on the task mapping problem. The main attention of this thesis has been focused on the mapping of a task graphs onto a clusterized parallel architecture, as illustrated in Figure 1.9.

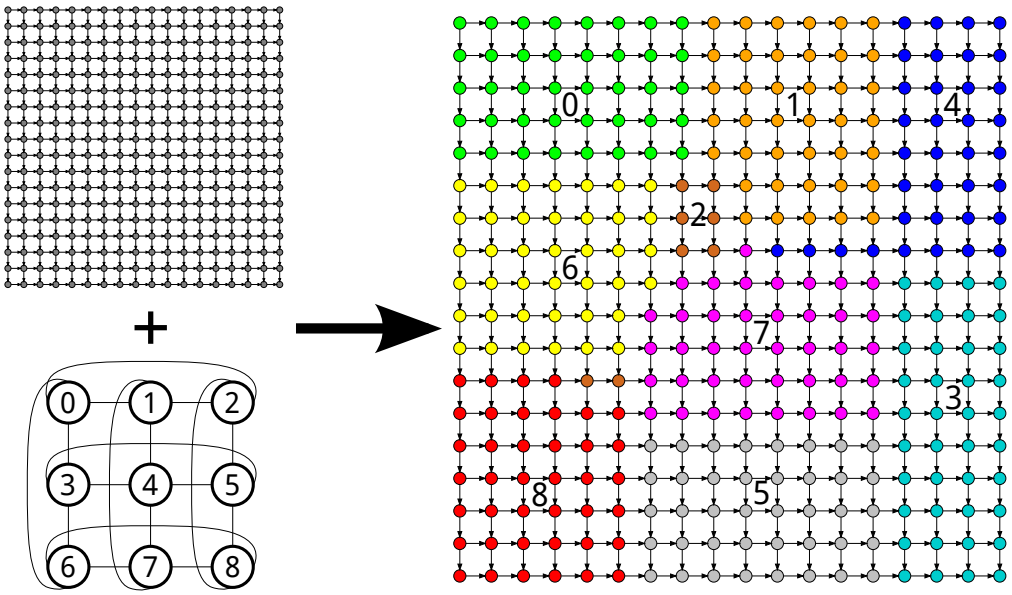


Figure 1.9: Mapping of a DPN on a parallel architecture. The grid on the upper left corner represents a task graph, the torus on the lowest left corner represents the target architecture. The graph on the right size represent the tasks graphs and colors indicates on which node the task is mapped. Each color corresponds to one node.

1.7 Formalization of the DPN mapping problem

Let us consider Figure 1.9. It corresponds to the mapping of a grid on a torus architecture. We want to know if this mapping allows the application, represented by the task graph, to achieve good run performance. This raises several questions: what is a good mapping? What is a bad mapping? How can we evaluate the mapping? Can the mapping be modeled while respecting the constraints?

This section will try to provide as many answers as possible. First, the mathematical mapping model able to evaluate the mapping is introduced.

Let T denote the set of tasks in the DPN and N the set of nodes or clusters. Let R denote the set of resources offered by the nodes, e.g., memory capacity, processing capability. Also, let w_{tr} denote the consumption of tasks t in resource r, $q_{tt'}$ denote the bandwidth between tasks $t \neq t'$ and $d_{nn'}$ denote the routing distance between nodes $n \neq n'$. Also, for the sake of simplicity and with a slight loss of generality, we assume that all nodes are identical and we denote by C_r the capacity of any of the nodes for resource r.

Given the variables

$$x_{tn} = \begin{cases} 1 & \text{iff task } t \text{ is assigned to node } n, \\ 0 & \text{otherwise,} \end{cases}$$

our DPN placing problem can then be expressed as the following mathematical program:

$$\begin{cases} \text{Minimize } \sum_{t\in T}\sum_{t'\neq t}\sum_{n\in N}\sum_{n\neq n'} x_{tn}x_{t'n'}q_{tt'}d_{nn'}, \\ \text{s. t.} \\ \sum_{n\in N} x_{tn} = 1 \ \forall t \in T, & (1.4) \\ \sum_{t\in T} w_{tr}x_{tn} \leq C_r \ \forall n \in N, r \in R, & (1.5) \\ x_{tn} \in \{0,1\} \ \forall t \in T, n \in N. \end{cases}$$

Constraints of type (1.4) simply express that each task must be assigned to one and only one node and constraints of type (1.5) require that the node capacity is not exceeded.

This generalized quadratic assignment problem (GQAP) is straightforwardly NP-hard in the strong sense, notably by restriction to the Node Capacitated Graph Partitioning Problem [51] (arbitrary network topology and bandwidths as well as equidistant nodes), to the Quadratic Assignment Problem (in the case where the capacity constraints allow to assign one and only one task per node and where the inter-node distance is arbitrary) as well as to the bin-packing problem. In the present thesis, we restrict the study to only one resource.

In terms of instance size, in our application context, we want to be able to map networks of over hundred thousands of tasks onto architectures having several hundreds of nodes. Such an order of magnitude rules out exact resolutions methods: the best known methods for the node capacitated graph partitioning problems are limited to graphs with a few hundreds of vertices, and the best known algorithms for the QAP are limited to instances of size around 128 [55]. Therefore, in our specific context, heuristic approaches are required to provide results for our problem on large graphs in reasonable time.

In this dissertation, the focus is set on the problem of mapping a dataflow process network onto a clusterized parallel microprocessor architecture composed of a number of nodes interconnected by an asynchronous packet network.

Target architectures are composed of SMPs with associated resources (CPU, memories, I/O). Each of the SMPs present in the architecture may have strictly identical characteristics. In that case, they are denoted as homogeneous. The architecture may also be made of different types of SMPs or different types of processors like accelerators. In that case, these architectures are denoted as heterogeneous.

In this thesis, the focus will be set on homogeneous architectures because it causes a small loss of generality. Step by step, starting with tasks graphs having the following features: tasks and edges are unitary weighted, two heuristics have been developed for these type of instances. Then, the research has been deepen in order to determine heuristics which presents more realistic applicative cases with non-unitary task and edge weights.

As of 2015, the biggest manycore architectures reached 1024 cores. Applications consist in ever growing sizes of data sets to tackle. During our research, we wanted a heuristic able to overcome this limit and which is able to produce results in a reasonable amount of time. This is why the focus has been set on the mapping of large task graphs (up to $2,000,000$ tasks) onto massively parallel architectures (up to 1024 SMPs).

1.8 A metric to evaluate the quality of solutions

For the evaluation of the quality of heuristics, exact methods or tight bounds are commonly used as a baseline. However, there is no exact method able to provide solutions for instances with hundreds of thousands of vertices and the design of tight enough bounds scaling at that level is still an open problem. The usual way is to compare solution values generated by heuristics on small instances to those of exact methods (like the one used for QAP). However, due to the large difference in orders of magnitude, there is no way to know whether the extrapolation to large instances will provide comparable results. This makes impossible to carry out comparisons with an exact method.

1.8.1 Approximation measure of Demange and Paschos

Demange and Paschos (D&P) provides an approximation scheme which allows to locate the quality of any solutions in an interval bounded by the best and the worst possible solution [41]. They introduced a notion of equivalence among optimization problems like maximal independent set problems and minimum size vertex cover problem. A vertex cover is the complement of an independent set. From an optimization point of view, these problems are very similar: once a value of the vertex cover problem is obtained, a simple subtraction allows one to obtain the solution of the independent set problem. However, the traditional approximation measure, which consists in computing the ratio of the value obtained of the traditional approximation algorithm $\mathscr{A}$ and the optimal value OPT, provides very different results and cannot be compared. D&P proved that, for any comparable optimization problems, the $\frac{\mathscr{A}}{OPT}$ ratio is not precise enough and does not respect equivalence principle between optimization problems.

They propose a metric able to solve the compatibility problem between qualitative measurements used in the estimation of performance of algorithms and to enforce the equivalence principle between optimization problems. For that, they introduced another variable. It represents the worst result obtained for the problem, denoted as Ω. Determining the configuration resulting to the worst result is not obvious then!

$$\rho(\mathscr{A}, I) = \frac{\mathscr{A}(I) - \Omega(I)}{OPT(I) - \Omega(I)}, \quad \rho(\Pi, \mathscr{A}) = \inf_{I \in \mathscr{I}(\Pi)} \rho(\mathscr{A}, I) \tag{1.6}$$

where:

- Π is an optimization problem,
- $\mathscr{I}(\Pi)$ is the set of instance of Π,
- I is one instance of $\mathscr{I}(\Pi)$, and
- $\rho(\mathscr{A}, I)$ and $\rho(\Pi, \mathscr{A})$ are differential approximation ratios.

If $\mathscr{A}(I) = OPT(I)$, then $\rho(\mathscr{A}, I) = 1$. If $\mathscr{A}(I) = \Omega(I)$, then $\rho(\mathscr{A}, I) = 0$ and if $\Omega = OPT(I)$, then the ratio is undefined. This ratio allows one to compare solution quality of an algorithm on two optimization problems. In [41], D&P show the effectiveness

of this differential approximative ratio on many NP-hard problems. Moreover, the use of the best and the worst values leads to locate the value of the solution in a bounded interval.

We want a metric able to evaluate the solutions of the heuristics we developed. Moreover, we do not have the optimal value and thus cannot approximate the optimal; this is why we cannot apply the D&P differential approximation ratio. Moreover, the idea is not to compare the effectiveness of our heuristic on many instances of several optimization problems but only one: the mapping problem. However, the fact of using the worst value in order to get comparative results depending of any type of optimization problems, gives us the basis for the design of a new approach. In this case, the worst possible value consists in determining the maximal value of the objective function defined in 1.7, by analogy, we chose to compare ourselves to objective function values obtained by means of random mappings. The main feature of this "worst" mapping remains in the fact that no characteristics, which are required in order to orient the mapping, are considered. This is why this approach can be considered as the "worst" in this case. On the contrary of the approach of D&P, this approach uses only two variables: the random value obtained by the mapping described in the next section and the results of the algorithm. However, the logic of evaluating how far the solution values of the evaluated algorithm are from random values is the basis logic used in the establishment of our random based metric.

1.8.2 Random-based approximation metric

We propose a comparison environment inspired by the differential approximation theory of Demange and Paschos. In this context, we compare solution values with those arising from a random mapping, which does neither take into account the task network architecture nor the target topology. The random mapping algorithm consists in sequentially placing each task on a uniformly randomly chosen node. Whenever there is not enough space in the selected node, another node is randomly chosen.

This random process is repeated several times. The average of all generated solution values is used as a comparison point with the heuristic solution value. The quality of a solution obtained from a heuristic can be expressed in terms of the ratio between the average of random solution values and the heuristic solution value.

Theorem 1. *The average cost value of all possible mappings is:*

$$v = \frac{1}{|N|^2} \sum_{n\in N} \sum_{n'\in N} d_{nn'} \sum_{t\in T} \sum_{t'\neq t} q_{tt'} \quad . \tag{1.7}$$

Proof.
$$\begin{aligned} v &= E(\sum_{t\in T} \sum_{t'\neq t} \sum_{n\in N} \sum_{n'\in N} x_{tn} x_{t'n'} q_{tt'} d_{nn'}) \\ &= \sum_{t\in T} \sum_{t'\neq t} q_{tt'} E(\sum_{n\in N} \sum_{n'\in N} x_{tn} x_{t'n'} d_{nn'})) \\ &= \sum_{t\in T} \sum_{t'\neq t} q_{tt'} \sum_{n\in N} \sum_{n'\in N} (\Pr(x_{tn}=1) \Pr(x_{t'n'}=1)) d_{nn'} \\ &= \sum_{t\in T} \sum_{t'\neq t} q_{tt'} \sum_{n\in N} \sum_{n'\in N} (\tfrac{1}{|N|} \times \tfrac{1}{|N|}) d_{nn'} \end{aligned}$$

$$= \sum_{t\in T}\sum_{t'\neq t} q_{tt'} \frac{1}{|N|^2} \sum_{n\in N}\sum_{n'\in N} d_{nn'}$$
$$= \frac{1}{|N|^2} \sum_{n\in N}\sum_{n'\in N} d_{nn'} \sum_{t\in T}\sum_{t'\neq t} q_{tt'} \qquad \square$$

Using this theorem, it is possible to directly compute the average cost value instead of generating random mappings.

1.9 Graph Theory Background

In this section, basis notions and terminologies of graph theory are defined. Moreover, the breadth-first search algorithm is also introduced because it is used in both heuristics developed in chapter 3. Moreover, a way to evaluate on which node a task should be mapped is also presented.

1.9.1 Some Definitions

A graph is denoted $G = (V, E)$ where V corresponds to the set of vertices (or tasks) of the graph, also denoted as $V(G)$, and E corresponds to the set of edges (which are the abstraction of communication channels of the task graph) of the same graph G also denoted as $E(G)$. $|V|$ and $|E|$ are respectively the number of tasks and edges in the graph.

A subgraph is denoted $G' = (V', E')$. It consists of a subset $V' \in V$ of vertices and a subset $E' \in E$ of edges constituted of all edges of E whose ends are in V'.

An edge $e_{ij} \in E(G)$ is incident to vertex i and vertex j which are the vertices at each end of the link (i, j). i and j are then considered **adjacent**.

A weight is an integer value which is attributed to either a vertex or an edge.

An oriented graph consists in a graph whose edges which have a direction. Oriented edges are commonly denoted as **arcs**. e_{ij} is the arc which starts from vertex i and goes to vertex j and is distinct from arc e_{ji}. The maximal number of arcs is defined as:

$$|E| \leq |V| \times (|V| - 1) \tag{1.8}$$

A non-oriented graph is a graph whose edges have no directions. This means $e_{ij} = e_{ji}$. The maximal number of edges is

$$|E| \leq \frac{|V| \times (|V| - 1)}{2} \tag{1.9}$$

In this dissertation, all task graphs are non-oriented.

A path p in a graph $G = (V, E)$, starting from a task $v_0 \in V(G)$, is defined as a sequence of vertices $(v_0, v_1, v_2, ..., v_k)$ connected to each other by a sequence of edges $(e_0, e_1, ...e_{k-1})$. A path is defined as *simple* if all vertices in the sequence are distinct. The **length** (or **distance**) of a path corresponds to the number of edges between v_0 and v_k if the graph is unweighted, or the sum of weights of the traversal edges if the graph is weighted.

A cycle is formed by a path $p = (v_0, v_1, v_2, ..., v_k)$ where $v_0 = v_k$. A simple cycle is a cycle where all vertices $v_0, v_1, .., v_{k-1}$ in the cycle are distinct.

A simple graph is a graph which has no cycles. It can be also denoted as a **tree**. All tasks graphs in this dissertation are simple graphs.

A connected graph contains at least one path between all distinct pairs $(v_i, v_j) \in V^2$.

The diameter corresponds to the shortest weighted simple path determined among all distinct pairs of tasks of the graph.

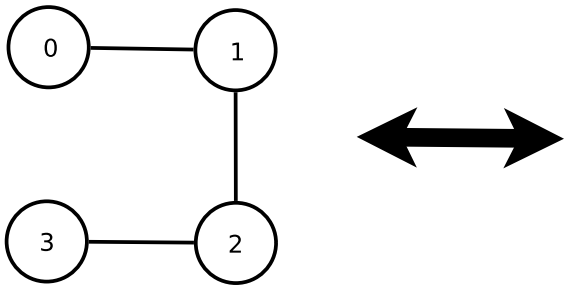
The eccentricity of a vertex $v \in V(G)$corresponds to the longest distance determined between v and all other vertices in $V(G)$.

The degree of a vertex $v \in V(G)$ is a measure denoted as $\delta(v)$ which corresponds to the number of edges ($e \in E(G)$) which are connected to v. The minimum degree $\delta(G)$ consists in the lowest degree value for each vertices and the maximum degree (often denoted by $\Delta(G)$) corresponds to the maximal degree value of all vertices.

The adjacency matrix of a graph $G = (V, E)$ corresponds to a $|V| \times |V|$ matrix M. For each pair of vertices $(i, j) \in V^2$, the m_{ij} coefficient value, at row i and column j in M is the weight of edge or arc $e_{ij} \in E$ and 0 otherwise. If G is unweighted, all edge/arc weights are set to one. As an example, Figure 1.10 shows a simple graph and its associated adjacency matrix.

The neighborhood of vertex v corresponds to all adjacent vertices of v. It is often denoted as $N_G(v)$.

Now that the required terminology of graph theory has been defined, the next step consists in describing the breadth-first search (or traversal) algorithm that we use. It is a graph traversal algorithm which is able to provide help in manipulating the locality of vertices of the graph. This is why it is the core of both mapping heuristics.



	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	0	1	0	1
3	0	0	1	0

Figure 1.10: A non-oriented simple fictitious graph with its associated adjacency matrix.

1.9.2 Breadth-First Traversal (BFT) algorithm

The two main graph traversal algorithm are Depth-first search and Breadth-first traversal (or search) (BFT). Only the Breadth-first traversal algorithm is detailed because it explores all neighbors of a starting vertex first, then the neighbors of the neighbors and so on. This approach allows to determine and to gather all unassigned vertices with the lowest distance from the starting task.

The French mathematician named Charles Tremaux was the first to create and to use this algorithm in order to solve maze problems [47],[182] in the 19^{th} century.

The algorithm uses a FIFO queue which is initialized with a single vertex v_0 which is the starting vertex. As long as the queue is non-empty, a vertex is pulled from the queue and all its neighbors that have not been explored yet are pushed into the queue. Since, for each explored vertex, all of its neighbors are tested for exploration, the complexity of BFT is $\mathcal{O}(E)$.

Now that all required graph theory notions are introduced, the first mapping heuristic is presented in the next session. This heuristic is the two phase mapping approach which computes an adapted version of the BFT in order to build connected subgraphs.

1.9.3 Notion of affinity

A problem to solve is how to evaluate to which node the subset of tasks has to be mapped, such that the local cost is minimized. This local cost should reflect how much the subset is worth when it is mapped on one particular node. For this purpose David *et al.* [37] introduced a valuation based on affinity for a polynomial method for splitting a graph into weakly interconnected subgraphs. It has been later used in many works from Stan [176] and Sirdey [174]. This valuation is used for the evaluation of a partitioning during a progressive construction approach.

The affinity computation consists in evaluating the affinity between two subsets. A subset consists in either a task, a group of tasks (like a subgraph) or a node. Let T_1, T_2 be two subsets of task set T from the DPN. The affinity computation $\alpha_{T_1T_2}$ between T_1

and T_2 is characterized by equation 1.10.

$$\alpha_{T_1T_2} = \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} q_{t_1t_2} \quad . \tag{1.10}$$

This valuation will serve as a source of inspiration and be adapted to the mapping problem.

1.10 Conclusion

In this introductory chapter, the context that surrounds our work has been presented. The emergence of new massively parallel embedded architectures and new applications which are able to be computed in parallel led to the necessity of developing new methods able to use those new platforms in the most effective way.

The introduction of a new programing language able to manage high parallelism allowed the setting of a new compilation process in which the mapping of a task graph generated by the compiler, with dependencies and resource occupation on parallel target architectures has an important position. This drove us to our research motivation because we wanted to map large parallel application onto these architectures.

The optimization problem related to the mapping problem is presented and the challenge raised by this problem has been detailed. In order to evaluate the quality of the heuristics which are presented in this thesis, we got inspired by a differential approximation ratio introduced by Demange and Paschos and developed a new metric based on random mappings.

In the next chapter, we will look deeper in the literature in order to enumerate all works that have already been cancel out on the mapping problem. The state of the art is split in two parts and, depending on the requirements presented in this chapter, is organized in order to locate the interest of this work in the current literature.

Chapter 2

State of the Art of the Mapping Problem

Contents

2.1 Introduction

The mapping problem has been formulated by Bokhari in 1981. He linked the mapping problem to graph isomorphism, bandwidth reduction and quadratic assignment problems. He also introduced the first mapping heuristic based on pairwise interchange [21] and made the suggestion that research in this "new" field should focus in the development of heuristics able to provide good solutions. This has been the starting point of the elaboration of many mapping heuristics.

In this chapter, the importance of having a good mapping method in order to place applications on target architectures is highlighted. A large number of mapping approaches for many types of constraints can be found. This makes an exhaustive listing difficult. This is the reason why it is necessary to find a classification which allows us to deal with the literature which surrounds the problem expressed in Subsection 1.7. Many classifications of heuristics can be found in the literature. The classification of one-phase mapping heuristics is based using [171]. Other classifications make a distinction

between iterative algorithms and greedy algorithms [5], [137]. Mapping is often performed by first partitioning a task graphs into partitions and then assigning partitions to the corresponding node. This is the reason why the state of the art of the partitioning problem and the quadratic assignment problem are explored. Then, the state of the art of the whole mapping process (partitioning and assignment) is explored in order to determine which problems are considered. Solvers can also be found in the literature. They are able to solve the partitioning and the mapping problems for large task graphs. For all cited articles, a short description of the mapping strategies are made. Once all articles are explained, a discussion about the state of the art is carried out. In addition, a more precise explanation of how massive parallel solvers perform mapping is done. All articles are summarized in a table which points out the size of task graphs of each application, the optimization problem constraints and also whether a target architecture is considered or not.

2.2 The Importance of Mapping Applications on Homogeneous Systems

In this dissertation, the interest will be raised on the SMP model. In this model, threads can be assigned or re-assigned to different processors depending on the needs and optimization constraints required by the application. Among several risks, are clashes of priority, deadlocks, data inconsistency, data starvation. MPSoC are designed to reduce these risks and the data flow model explained in Section 1.5 also helps to avoid them. However, the design of MPSoC able to process large and complex applications is a difficult research field [131] but we are not going to focus on it [132].

In order to optimize the use of these systems, an application should be mapped wisely onto them. The difficulty of this wise placement lies in how to get the best suitable mapping. This implies to identify the associated algorithmic problems and to solve them. In other words, the resource allocation problem is one of the biggest issue: how many cores should be used by the application and how to map tasks wisely in order to maximize system performance? Another problem is the quadratic assignment problem, which is identified as the general problem of mapping applications onto several cores. Moreover, it is a NP-hard problem [66].

For mapping applications, one existing approach consists in partitioning it into many processing tasks. The transformation of an application into a task graph is done using the transformation process cited in Section 1.5. Powerful partitioning solvers like Metis [95] or Scotch [145] (or their parallel versions Parmetis [100] or PT-Scotch [142]) are able to partition any task graph onto a target architecture. Task mapping is a process of assigning and ordering the tasks and their communications on cores depending on available resources and rules of optimization and constraints. In our case, we are dealing with capacity constraints on each core. User demands in performance for each application have to be fulfilled. This means applications should be mapped in a way that no better mapping which can provide better response to the demand can be found. We can define a model which suits the most to the user demands knowing that it consists of an

interpretation. Mathematical optimality of models of a solution may not be a necessary condition for the satisfaction of user demands. These demand can be satisfied in terms of quality of solution compare to resolution time. In order to map small applications (up to 120 tasks), exact methods can directly be used in order to obtain an optimal mapping. In configurations where the number of tasks is more important, heuristics or parallel approaches provide a mapping which takes the objective function into account. In addition to the quality requirement, heuristics must also exhibit flexibility, robustness and experimentally provide seemingly good solutions for any type of applications. As can be expected, when the application is big, the mapping phase is hard to be well performed.

The mapping problem which is raised by answering to optimization's constraints on modern embedded systems has become critical. Due to the existence of several optimization constraints, nowadays, no global mapping method is able to provide an optimal mapping regardless of any constraints [133]. This aspect led to the development of a large number of heuristics, exact methods or parallel approaches, in order to provide efficient response depending on the considered constraints. In this state of the art, we propose a classification of the most recent and efficient mapping methods.

2.3 Partitioning problems

The partitioning problem consists in splitting a set of vertices of a graph into smaller subsets of vertices while minimizing the sum of the costs on all egdes cut. Kernighan and Lin were among the firsts to address this problem [103]. They showed that the number of possible partitions is

$$\frac{1}{k!}\binom{n}{p}\binom{n-p}{p}....\binom{2p}{p}\binom{p}{p} \tag{2.1}$$

where:

- n is the number of vertices,
- k is the number of subsets of size p, where $kp = n$.

making an exhaustive search for the solution impossible when the number of vertices starts becomes large. Moreover, many form of partitioning rules can be applied, like partitioning graphs into triangular configurations, perfect matching [163], isomorphic subgraphs [105], Hamiltonian subgraphs [185], forests [66] or cliques [91]. All these partitioning problems are NP-Hard [66]. In this dissertation, the aim of the partitioning consists in minimizing communication costs by taking into account the bandwidth between tasks, times the distance between partitions or nodes.

The first partitioning approach which leads to local optimal values is that of recursive bipartitioning algorithms. In addition, they are also used in another form of partitioning also detailed below: the multilevel approach. Last, partitioning solvers able to deal with several millions of tasks are detailed.

2.3.1 Bipartitioning algorithms

Kernighan-Lin (KL) This algorithm [103] is one of the most effective approach able to determine locally optimal partitions if it starts with a good initial partition. The KL algorithm incrementally swaps vertices among parts of a bisection in an attempt to find two disjoint subsets of equal size with the lowest edge-cut of the partitioning. It is performed until a local minimum is found. In the case of multilevel recursive bisection algorithms, KL refinement becomes very efficient as the initial partitioning available at each successive uncoarsening level is already a good partition.

Fidduccia-Mattheyses (FM) This algorithm [52] is a "linear time heuristic for improving network partitions". It includes additional features to KL like a reduction of net-cut costs, it takes into consideration vertices weights and is able to handle unbalanced partitions.

These algorithms are mostly used for load balancing problems and are used in many multilevel approaches. They can be used for recursive bisection heuristics.

2.3.2 Multilevel approaches

As explained just above, finding the best partition in order to solve the mapping problem leads to solve another NP-hard problem. One class of graph partitioning algorithm consists in first collapsing vertices and edges in order to build a smaller graph (coarsening phase), then to partition the smaller graph (partitioning or mapping) and last to spread it to construct a partition for the original graph [12], [84]:

1. **Coarsening phase**: during this phase, a sequence of smaller graphs is constructed from the original graph. For each sequence level, a set of vertices V' is combined together in order to form a new vertex. The weight of this vertex corresponds to the sum of all weights of vertices of V'. In order to preserve the graph connectivity, edges whose ends are in V' are merged. Most of the time, an *Heavy Edge matching* algorithm is used in order to perform this operation [15] [32] [96].

2. **Initial partitioning or mapping**: This phase is a simple assignment of the groups to the processors. Many partitioning or mapping heuristics present in the literature can be used for this phase depending on problem constraints.

3. **Uncoarsening or refinement phase**: During this phase, the partition built in the previous phase is projected back to the original graph by going through the sequence of graphs built during the coarsening phase. At each level of the sequence, local refinement heuristics are performed in order to improve quality of partitions. Aforementioned heuristics Kernighan-Lin [103] or Fidduccia-Mattheyses [52] are mostly used for this phase.

The main difficulty consists in finding efficient ways to coarse "and uncoarse" the task graphs. It has been subject to many works whose results can be found in [32], [96], [99], [60].

The multilevel approach is the core of many partitioning solvers which are detailed below.

2.4 Quadratic Assignment Problems (QAP)

The assignment problem is introduced by Koopmans and Beckmann in 1957 in order to locate indivisible economical activities [109]. The assignment problem terminology is applied to our problem. Consider the problem of assigning n tasks to n nodes. The task (f_{ij}) and node $(d_{\varphi(i)\varphi(j)})$ matrices are symmetric, nonnegative and have zeros in the diagonal. The cost assignment matrix $(b_{i\varphi(i)})$ is nonnegative. The mathematical model of Koopmans and Beckmann is represented by the following equation:

$$\min_{\varphi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} + \sum_{i=1}^{n} b_{i\varphi(i)} \ , \tag{2.2}$$

where

- f_{ij} is the flow between two tasks,
- d_{kl} is the distance between two nodes,
- n the number of tasks and nodes,
- b_{ik} is the cost of placing task i on node k,
- $f_{ij} d_{\varphi(i)\varphi(j)}$ is the cost of assigning task i to node $\varphi(i)$ and task j to node $\varphi(j)$, and
- $\varphi \in S_n$ corresponds to the set of all permutations $\varphi : N \to N$.

Many combinatorial optimization problems can be formulated using Equation 2.2 by adapting the coefficient matrix [27]. The focus is set on the linear arrangement problem which is proven as NP-Hard by Garey and Johnson [66]. For linear assignment problems, the following equation models the problem to solve:

$$\min \sum_{i,j=1}^{n} \sum_{k,l=1}^{n} f_{ij} d_{k,l} x_{ik} x_{jp} \ , \tag{2.3}$$

where:

- f_{ij} is the flow between two tasks,
- d_{kl} is the distance between two nodes, and
- n the number of tasks and nodes.

Equation 2.3 is very similar to the model presented in Section 1.7.

Loiola also proposes a survey which covers many heuristics able to solve QAP problems [118]. Moreover, Hahn was able to find lower bounds for this problem using the Hungarian method [74] and the branch and bound method in order to find optimal assignments [75], [160].

Many problems are derived from the QAP. The most similar problem to our mapping model is the Generalized Quadratic Assignment Problem (GQAP). Hahn *et. al.* propose many exact methods or heuristics able to solve or approximate it [75]. However, these approaches are limited by sizes of instances up to several hundreds of tasks.

Sirdey used a partitioning based on an affinity-based GRASP in order to generate as many partitions as the number of processors. Then, a simulated annealing heuristic is used in order to solve the associated QAP problem [174].

Heuristics developed for the resolution of the partitioning problems focus on minimizing edge cut or on enforcing load balance constraints. However, generated partitions do not take into account a notion of distance between partitions. The application of partitioning heuristics on the mapping problem leads to mappings which are suboptimal.

The assignment problem focuses on assigning a number of subsets of tasks on an equivalent number of nodes. Despite of the fact that the mathematical model is similar, we want to map n tasks onto m processors. Heuristics able to solve the GQAP problems are able to deal with small instances but they are not scalable. This is the reason why, for instances larger than several hundreds of tasks, a partitioning phase is required in order to reduce the number of subsets of tasks to map.

2.5 Mapping problems

Two static mapping approaches are presented in this section. First, a two-phasess mapping approach which consists in partitioning the tasks graph before placing partitions on SMP nodes. This process is performed sequentially. Second, a one-phase mapping approach which directly assign tasks to nodes.

The two types of mappings are illustrated by Figure 2.1.

2.5.1 Two-phases mapping heuristics

Alternatively, in two phases mapping, task graphs are first partitioned into smaller graphs according to various sets of rules and algorithms. This first problem is a partitioning problem. Once partitions are determined, they are mapped onto the corresponding processing elements (or processors). The partitioning approaches are split into four categories: partitioning strategies, multilevel algorithms, clustering algorithms and spectral approaches. In this thesis, despite of the fact that partitioning and clustering are the same problem, a distinction is performed: partitioning splits a graph into several subgraph. It is a top-down approach. In contrast, clustering gathers tasks together. This is a bottom-up approach. The principle of partitioning and multilevel are detailed in Section 2.3. In clustering algorithms, all tasks are gathered into several clusters using a well defined and

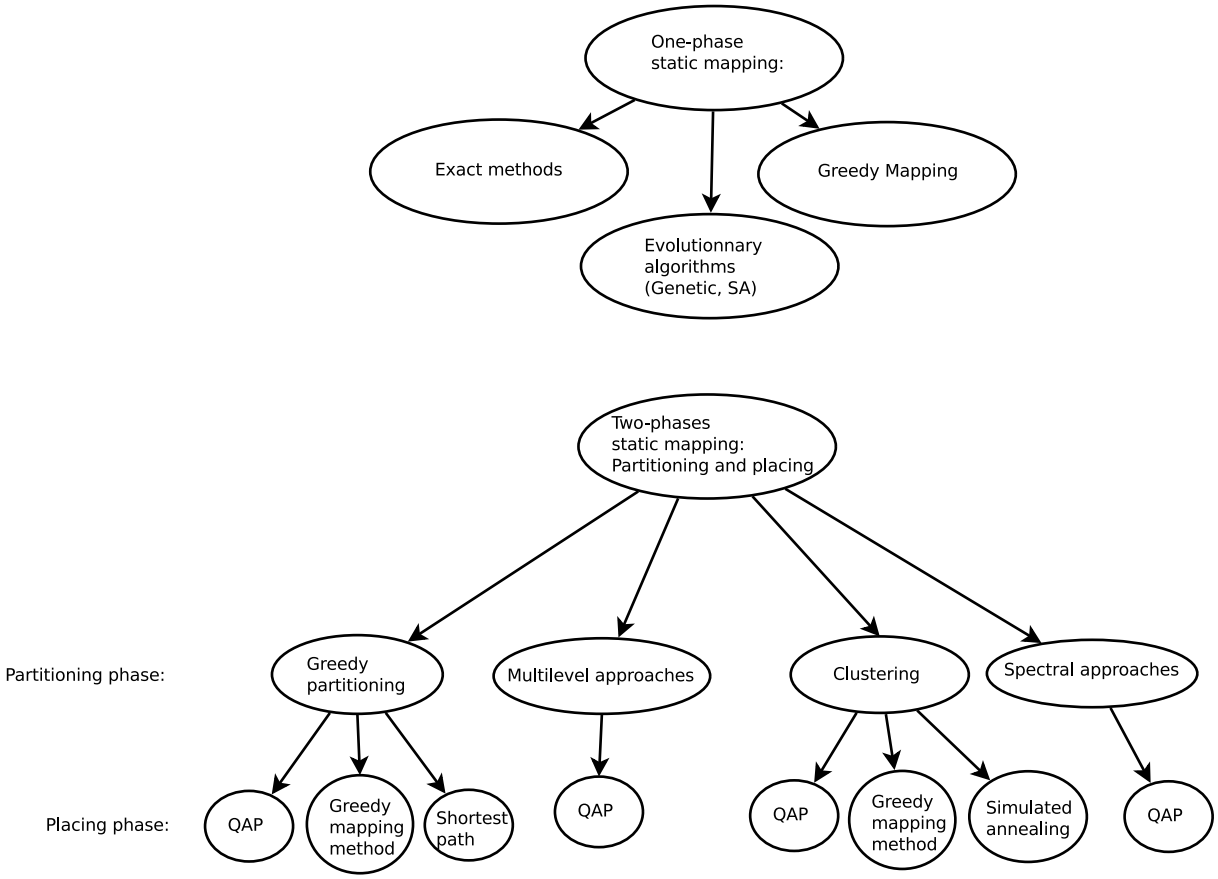


Figure 2.1: The state of the art has been filtered using the above personal criteria. Placing heuristics are not exhaustive, only those which appeared frequently are cited

adapted set of clustering rules. Once the clusters are formed, each cluster is mapped onto the corresponding processing elements. In the spectral approaches, the Lagrangian matrix (which has no relation with the Lagrangian operation cited in Section 1.6.2) is computed and analyzed. Depending on mathematical rules which are set up for this purpose and directly applied to the matrix, tasks are gathered into sub-matrices and mapped More details about spectral approaches functioning are provided later in this dissertation.

Partitioning

A k-way graph partition corresponds to a way to divide a graph $G = (V, E)$ into k smaller sets of vertices using specific properties, as explained in Section 2.3. This means finding the best partition which satisfies several criteria is difficult for most graphs. The reason why, is that the exploration of all possible and feasible partitions of G can not be done in reasonable time.

Several solvers are able to provide good partitions for large task graphs under load balancing constraints, as explained in Section 2.6. Despite the efficiency of these solvers, many other approaches provide comparable or, depending on the problem to solve, better results than the solvers, which are more generalistic. Kernighan-Lin [103] and Fidduccia-Mattheyses [52] are some of the most famous bi-partitioning heuristics, as explained in Subsection 2.3. Battiti for instance did many works in clustering, partitioning and the multilevel partitioning fields like randomized greedy approach for the partitioning problem [14]. Kirkpatrick [106], on his side, was able to establish a partitioning heuristic based on simulated annealing. In the field of evolutionary algorithms, genetic algorithms also lead to interesting results [26]. But these methods are only the first part of the mapping. Once the partitioning is determined, either the obtained partition can directly be mapped onto a node or a greedy heuristic is processed in order to find the best way to assign partitions to nodes.

The mapping approach illustrated by [158] uses a two-step procedure. The first generates an initial "nearest-neighbor" mapping which is commonly used in many two-phases mappings. In the initial mapping, the vertices of the finite element graph are grouped into nodes and mapped onto processors. Two vertices that share an edge are assigned either to the same processor or to neighboring processors. The second step uses a boundary refinement procedure which tries to equalize the computational loads on processors by refining the boundaries obtained by the initial mapping.

Another approach, which is used in networked Clouds [116], partitions the user requests using iterated local search by a random walk in the space of local optima. Once the requests are partitioned, the networked cloud mapping is performed. It is done in two phases: node mapping and link mapping. The node mapping is transformed into a flow allocation problem solved by linear programming and the link mapping is solved using the shortest path algorithm.

In Agarwal's approach [2], the partitioning is first performed by the Metis [95] solver. Once the task graph have been split into sets of p tasks, where p the number of processors, the more heavily communicating tasks are mapped onto nearby processors. Sometimes, it may happen that the allocation of a task to any processor does not affect the global

mapping cost. A notion of critical task is defined by approximating the fact that if a task is not placed at the current cycle, it may be mapped onto a random processor later.

However, one must not forget that the graph partitioning problem is as difficult as the task mapping problem. This means that a direct partitioning of a large task graph may present the same difficulties than directly finding a mapping method. Clearly if the number of tasks is too important with respect to the heuristic complexity, finding a satisfactory partition in a reasonable amount of time might become impossible. This is why, a new partitioning approach, called multilevel partitioning, has been introduced in order to avoid the aforementioned problem.

Clustering algorithms

Clustering-based approaches have a similar logic as partitioning approaches. That is to say, task graph are split into clusters, then clusters are mapped on processors. If the system resources allow it, clusters can also be merged into new clusters [174]. Clustering algorithms are massively used in the domain of big data, machine learning and problems like the traveling salesman problem. Clustering algorithms are particularly efficient in dividing sets of data into a defined number of clusters.

Xu *et al.* write a very complete and diversified survey about clustering algorithms [197]. In his survey, many fields of research are detailed. Many types of clustering can be found and have their corresponding denotation, like *linkage clustering*, which is a clustering problem which is very similar to the mapping problem. Jain *et al.* assume that linkage clustering can be related to the search of maximal connected subgraphs [89]. Moreover, Karypis and Kumar also developed an approach based on hierarchical clustering for the partitioning and mapping domain [93] in which the notion of k-nearest neighbor mapping [62], [138], [155] is introduced. At the first step, the connected graph is divided into a set of clusters using a minimal edge cut algorithm. Another work from Karypis and Kumar introduced the Chameleon algorithm able to find several clusters among a data set [93]. It is applied on sparse-graph representations which allows this algorithm to scale on large data sets. Using communication between task graph and distance of the clusters, Chameleon merges all small clusters until a final solution is obtained. A similar approach has also been developed by Sadayappan *et al.* [157] for planar graphs. Chen and Lo [31] later used a two-phases heuristic which leads to the same results as Chameleon.

Clustering approaches are different approaches than partitioning approaches but are able to be used in the mapping problem and to solve problems of up to thousands of tasks.

Another method is based on progressive construction for the multi-resource Node Capacitated Graph Partitioning Problem [174]. The method consists of two phases: a partitioning phase and a mapping phase. The partitioning phase is a GRASP approach [48]. It is an affinity-based randomized iterative process which creates a partition of the tasks graph. The second part of the algorithm consists in a simulated-annealing-based quadratic assignment problem (QAP) heuristic, which assigns one partition of the task graph to each of the SMPs. This first method is fast and is suitable to the

early development cycle, where the programmer needs fast feedback from the compilation toolchain and does not focus on the quality of the mapping.

Spectral bisection approaches

Spectral bisection uses matrix operations in order to decide how the graph is partitioned. The second lowest eigenvectors λ_1 of the Laplacian matrix of the graph is computed in order to divide the task graph in two parts. Given a graph $G = (V, E)$, the Laplacian matrix $L_{n\times n}$ is computed using the degree matrix (D):

$$L_{i,j} = \begin{cases} deg(v_i) \text{ if } i = j \\ -1 \text{ if } i \neq j \end{cases} \tag{2.4}$$

of all vertices and the adjacency matrix (A) as shown in the following equation:

$$L = D - A \ . \tag{2.5}$$

Because of the semi definite positiveness of L, eigenvectors are increasingly ordered: $\lambda_0 \leq \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$. Usually, the first eigenvector has a zero value which is why the second eigenvalue is selected. This approach has been highlighted by Fiedler [53], [54]. Moreover, Fiedler's theorem guaranties the connectivity of the two generated task graphs. Using the same logic, a notion of quadrisection has been introduced, allowing to divide the graph into 4 subsets at once. This operation is performed using the second and third lowest eigenvectors of the Laplacian matrix. These mathematical operations make it simpler to divide the task graph into several dimensional partitions [83].

Bui, otherwise, introduced an approach able to find the optimal (minimal) bisection for all d ($d \geq 3$) regular simple graphs with $2n$ nodes [25]. A d regular graph is a graph where each vertex has the same number d of neighbors. Unfortunately, even if the number of tasks reaches thousands of tasks, it does not scale to higher orders of magnitude.

Spectral bisection is able to find very good partitions. In some cases, it is able to find the optimal value but this ability cannot be generalized. Moreover, the mathematical computation lasts too much and when the number of tasks starts to be consequent, it appears not to be able to find any solution. Once the partitions are defined, they are assigned on the corresponding processors without taking into account the target topology.

Multilevel approaches

We have already defined multilevel approaches in Subsection 2.3.2. Several multilevel approaches which are used for the mapping problem are listed below.

In an approach by Barnard and Simon [12], a mix of recursive spectral bisection and multilevel implementation is used in the mapping of unstructured meshes of task graphs on processors. The aim is to minimize communication while preserving load balance in the system.

Battiti, on his side, uses a greedy heuristic in the coarsening phase and the tabu search adapted to Kernighan-Lin [15] which allows to find a partition with the minimum possible cut that the heuristic can compute.

2.5.2 One-phase mapping heuristics

In the one-phase mapping, the task graph is directly mapped onto the target architecture. Two types of mappings can be found: dynamic and static mappings. In the dynamic mapping context, depending the data set given as input in the application, tasks are created at the execution of the application and they are processed once. There is no global vision of all tasks in the system. This mapping is denoted as *on-line* mapping. An advantage of this mapping is the ability to manage in real time the resources of the system and to avoid any defective part of the MPSoC. Methods like adaptive work stealing or on-line scheduling are massively used. In contrast static mapping is an *off-line* mapping. Tasks are defined by the program and created during the compilation phase as explained in Subsection 1.6.3. The execution is periodic and can be performed as many times as possible. Hence, it has a global view of the system. This facilitates the use of system resources. This mapping is performed in the compilation phase.

Many mapping methodologies in the literature fall under static mapping. While analyzing the whole mapping state of the art, it appears that mapping techniques are used in order to map applications on either heterogeneous or homogeneous MPSoCs. The focus is set on mapping methods for homogeneous MPSoC.

Many elements in the literature focus on energy consumption, reliability, temperature and performance (execution time, delay, latency, throughput, etc.) [171]. The challenge which is faced in this thesis consists in finding an approach or method which runs fast on large instances and able to provide good mapping quality, that is to say a mapping value as close as the minimal mapping value as defined in Subsection 1.7. This led us to consider only article which are dealing with performance. Figure 2.2 illustrates the decision tree which summarizes how we sorted the state of the art of one-phase mapping methods. It allows us to filter the state of the art which includes numerous works and to focus only on performance-oriented mapping method, in terms of computational time and solution quality.

Exact methods

Exact methods are able to reach optimal values for any small-sized algorithmic problems (depending on the problem, this size is often around 120 tasks) in a reasonable amount of time. This is due to the fact for small-sized instances, the complexity of the algorithm remains quite high. However, if slightly increasing problem size may induce a substantial increase of the complexity of the problem, under these circumstances, it takes a higher amount of time for determining the optimal solution. For the mapping problem, the optimal search consists in finding the mapping which has the lowest cost. Cost computation varies depending on the problem to solve. Some exact methods use branch and bound algorithms [168], [170] or dynamic programming [77], [154].

Depending on the number of processing elements, the problem can be reduced to a polynomial approach, especially if the number of processors is 2. Some partitioning or assignment exact method can be used as a mapping approach. Stone showed a way to find a near-optimal mapping based on assignment algorithm on a 2-processor system in a

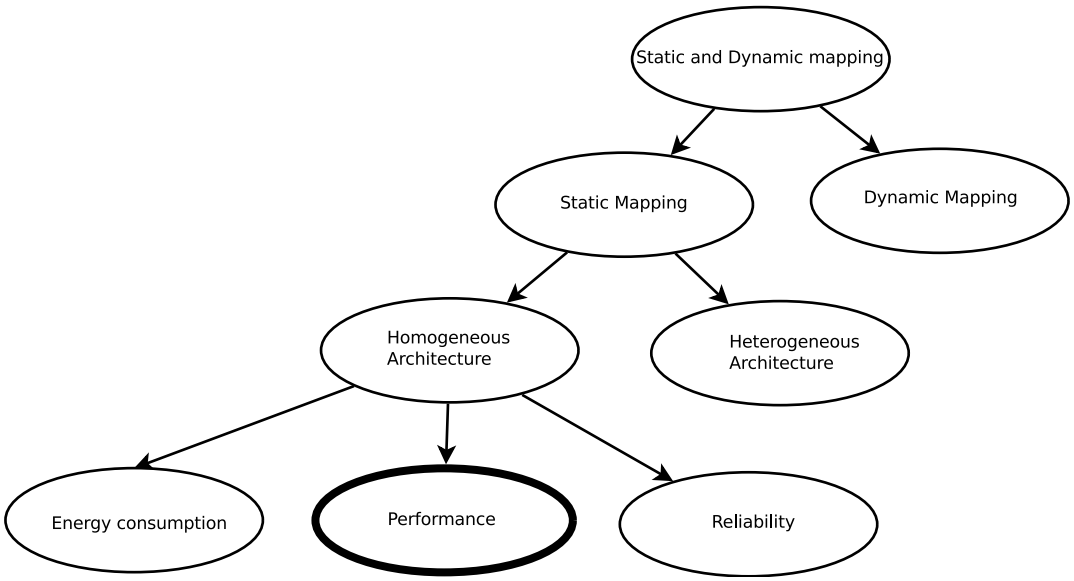


Figure 2.2: Decision tree which shows how the state of the art of one-phase mapping is organized. The focus is set on the performance of static mapping heuristics which are developed for homogeneous target architectures. This classification is inspired by a survey of Singh *et al.* [171].

polynomial time [179]. Later, Lo [117] managed to find a way to generalize it on several processors systems but the approach lacks of robustness.

Evolutionary algorithms

The domain of stochastic optimization algorithms includes evolutionary algorithms. For instance, simulated annealing (SA) [106] (which has been applied to many mapping and scheduling problems) or genetic algorithms [164] belong to this class of algorithms. These algorithms are considered as meta-heuristics [140] and are widely used for hard optimization problems. The main difficulty consists in finding adequate parameters making these heuristics able to adapt to the current mapping problem [23].

Hu and Marculescu [87] developed a deadlock-free deterministic mapping and routing method based on simulated annealing (SA) and branch and bound approaches. 3000 random mappings are generated. In order to determine the best mapping, SA is performed on these mappings. This mapping minimizes the energy consumption under specified performance constraints.

Galea *et al.*'s approach [65] consists in a parallel simulated annealing method. This is a single-phase heuristic which directly assigns tasks to the SMPs. It provides results which are better but takes considerably more time than Sirdey *et.al.* [174] even though parallelism allows to drastically reduce execution time.

Erbas *et al.* proposed a genetic algorithm-based approach [45] which is used in their Sesame software framework. In order to solve large instances of the mapping problem, multiobjective evolutionary algorithms (MOEAs) are applied. Optimization criteria in-

clude minimum processing time, minimization of power consumption and minimization of the total cost of the solution.

Orsila *et al.* proposed a method based on simultaneous optimization of execution time and memory consumption [139]. Applications are mapped using simulated annealing. In addition, one major strength of their approach is the establishment of an automated parameter selection able to optimize the selection of parameters used for mapping sets of tasks.

Marcon *et al.* compared several algorithms [130] to the SA-based approach they developed in earlier works [128], [129]. Many of the presented algorithms focus on low energy consumption but their SA approach, as well as Kreutz *et al.* [111] tabu search also focuses on computation time. Marcon *et al.* also introduced two greedy heuristics: largest communication first (LCF), which produces mappings using priority rules. The most communicating modules are processed before the least communicating modules. Another approach denoted as greedy incremental (GI) starts with an initial mapping. A position in the NoC is selected as "pivot" of evaluation. This "pivot" is used as a starting point for a local search procedure in order to find its best interchangeable element. The complexity of this algorithm is $\mathcal{O}(n^2)$, where n corresponds to the number of n-tile NoC. A tile contains a router and a module of a NoC placed inside a limited region on an IC. In this work, it appears that LCF provides good execution time while sacrificing energy gain or energy saving. A good compromise between energy and execution time is performed by GI and HSM. A mixed approach which combines LCF and SA also provides results comparable to those of GI and HSM.

General mapping heuristics

In an approach by Manolache *et al.* [127], a solution space S is built where each point of this space represents a special configuration which fulfills the problem constraints. Once it is formed, a tabu search-based exploration strategy is performed in order to find the best result.

Ruggiero *et al.* have a more simpler approach: an integer programming method combined with constraint programming allows to speed up the execution [156].

Bonfietti *et al.* explore an approach which deals with allocating and scheduling SDF subject to minimum throughput constraints [22]. The heuristic uses constraints programming in order to solve the allocation problem. Moreover, it also apply optimization techniques like minimizing throughput bound tightness in order to reduce the number of feasible solutions.

Most mapping strategies are using simulated annealing or, more generally speaking evolutionary algorithms and are able to find a solution value which is assumed to be close to the optimal values. Other more general approaches use strategies which allow to explore the solution space efficiently. However, the problems which are solved are neither the same than the one addressed by this work nor are able to be applied to large task graphs.

Of course, more mapping methods can be cited but their corresponding features start to be very different to ours. The previously cited approaches are very close to our re-

quirements.

2.6 Solvers

In this section, several solvers able to perform partitioning or mapping operations are introduced.

2.6.1 Metis, Parmetis, hMetis, kMetis

Metis has been developed by Karypis and Kumar. The first version has been introduced by [95] in 1995. It is more a partitioning approach which can be generalized to a mapping approach. The tool has been improved from 1995 until 1999 with the following researches: [94] deepens the multilevel strategy, [99] applies Metis to irregular graphs, [98] adapts Metis to multi-constraint partitioning. In 2003, Karypis alone adapted Metis for multi-constraint mesh partitioning [92]. Metis then has been added a low power feature [1] and the last improvement of kMetis appears in 2013 from a joint work of LaSalle and Karypis where some heuristics in the approach were adapted for parallelism [112]. Metis has two extended versions denoted as parMetis, which is the parallel version of Metis [100] and hMetis, which is specialized for hypergraphs [97].

Metis is able to partition several tens of million tasks on 256 partitions depending on load balancing constraints. Depending on how the task graph and the constraints are organized, the obtained partitions may be mapped onto processors using a greedy or an iterative heuristic. It has not been designed for mapping purpose, which is why it does not take into account a target topology during the partitioning phase.

2.6.2 Scotch and PT-Scotch

Another solver which presents equivalent performance than Metis is named Scotch. On the contrary of Metis which focuses on partitioning problems, Scotch focuses on the mapping problem. Scotch is based on the thesis of Pellegrini [143] and has been released by him in 1994 [145]. Sparse matrix ordering has then been added [146] in 1997 and completed with hybrid nested dissection [147]. Many features has been added like native mesh ordering and the use of a genetic algorithm for scalable parallel partitioning. A joint work of Chevalier and Pellegrini [32] gave birth to PT-Scotch which is the parallel version of Scotch [142]. One additional aspect to consider consists in its ability to map large task graphs overs hundreds of millions of tasks,

The Scotch project is split in two parts: the first part focuses on the static mapping. The second focus on the ordering of sparse matrices. This dissertation is about the mapping problem and the interest will only be laid on this aspect. Scotch solves the mapping problem using a dual recursive bi-partitioning algorithm [145], [141]. Several graph bi-partitioning heuristics have also been studied and can be found as black boxes in this solver. In order to solve the mapping problem, Pellegrini *et al.* defined a cost function using following requirements: minimization of communication cost under the constraints

of maintaining the load balance within an acceptable tolerance. It is expressed by the following equation:

$$f_C(\tau_{S,R}, \rho_{S,T}) \stackrel{def}{=} \sum_{e_S \in E(S)} w_S(e_S)|\rho_{S,T}(e_S)| \ . \tag{2.6}$$

f_C represents the communication cost function. It consists in the sum of all edge's dilatation $|\rho_{S,T}(e_S)|$ times their corresponding weights $w_S(e_S)$. S corresponds to the application to map and T corresponds to the target architecture. A communication channel in T can be composed by several communication channels of S. $\rho_{S,T}(e_S) = \{e_T^1, e_T^1, e_T^2, .., e_T^n\}$ indicates if edge e_S is routed on link e_T^1 or e_T^2 or e_T^n. $w_S(v_S)$ or $w_S(e_S)$ [142].

The load balancing can be defined by μ_{map} which is the average load per computational power unit represented by:

$$\mu_{map} \stackrel{def}{=} \frac{\sum_{v_S \in V(S)} w_S(v_S) v_T \in V(T)}{w_T(v_T)} \ . \tag{2.7}$$

The μ_{map} value is used to define an imbalance ratio to which the user provides an upper limit which must be enforced during the solution process.

Scotch, like Chaco [82] or Metis [95] also uses multilevel strategies/ One major difference between Scotch and other solvers is brought by a bi-partitioning heuristic denoted as Dual Recursive Bi-partitioning algorithm (DRBA). In this approach, for each level of the recursive algorithm, the set of processors is split in two and the set of tasks is also split in two. Once the size of the processor set is 1, the corresponding subset of tasks is mapped to it. During the bi-partitioning phase, one important aspect consists in maintaining the load balancing property. This leads to a partial communication cost function. Once the mapping is done, it is important to maintain the mapping quality or to improve it while the different subsets of tasks are merged. Scotch proposes several heuristics used as black boxes which allows to reduce mapping deterioration. Several heuristic are available like Band [33], Diffusion [144], Fidduccia-Mattheyses [52], Gibbs-Poole-Stockmeyer [68], Grasp (proposed by Karypis and Kumar in Metis), or more classical multilevel approaches expressed in Subsection 2.3.2.

2.6.3 Jostle: Parallel Multi-Level Graph Partitioning Software

Jostle is a multi-level graph partitioning software package which is dealing with graph partitioning problem under load balancing constraints [193]. The first version was developed in 1998 [190] by Walshaw and Cross and was later improve in order to solve several partitioning problems like heterogeneous communication networks [192] or optimization of domain shape [194].

This solver is able to map millions of tasks on hundreds of processors in several hundreds of seconds. In order to gain these performance, the solver is using parallel multilevel partitioning methods. First, the multilevel framework is parallelized, then several parallel approaches are used for the refinement phase [191].

They compared their results to Metis in terms of cut-edge weight and execution time. Jostle got equivalent results on cut-edge weights. However, on 16 processor, it runs 1.08 to 2.42 times faster than Metis but, when the number of processor is 128, it runs 0.54 to 1.11 faster than Metis.

Jostle can also be used for mapping purposes [192]. The aim of the mapping consists in minimizing the cut-weight costs while balancing the load or vertex weight in each nodes and taking into account the target topology. In this method, Walshaw and Cross are able to map millions of tasks on several tens of processors. Quality metrics which are used in order to evaluate the mapping are cut-weights, network costs, average dilation and average path length/unweighted dilation.

2.6.4 Other solvers

Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services Zoltan is a toolkit of parallel combinatorial algorithms which are dealing with load balancing partitioning, ordering and coloring problems [42] and also data migrations. This toolkit includes parallel solvers like ParMetis, PaToH and PT-Scotch. However, this solver does not deal with mapping or capacity constraints assignment problems.

PaToH: Partitioning Tool for Hypergraph This solver is used for partitioning hypergraph using load balancing constraints [29]. It uses the recursive bipartition and the multilevel principle in order to get the required number of partitions. However, this solver only focus on the computation of partitions and is not mapping oriented. The quality of the partitions are equivalent of that of Metis. Moreover, it is purely sequential and is one of the fastest for hypergraph. However, despite of the sequential aspect, parallel approaches runs faster.

KaHiP: Karlsruhe High Quality Partitioning KaHiP is a multilevel graph partitioning solver which mixes global search strategies from multigrid linear solvers to max-flow min-cut computations. This solver also solves the partitioning problem using load balancing constraints. They compare the solution quality of their approach to Jostle using instances of Walshaw [195] (which are also used in this thesis). KaHiP manages to get slightly better results than Jostle depending on the instances. However, they are only focused on the partitioning problem and not the mapping problem. [161]

Other solvers Many other solvers can be found. All of them are solving problems with load balancing constraints. Some are considering multilevel algorithm for Hypergraph like Parkway [184], other are reconsidering the *aspect ratio* of partitions like Party [148], or are oriented towards solving the edge-cut problem like DiBaP [135].

The main solvers have been detailed. The attention has been put on the fact that solvers, despite of the fact of being able to map hundreds of million tasks, mainly depend on load balancing constraints.

2.7 Discussion

In this section, a synopsis of the literature around the mapping problem is detailed and will enable a positioning of this thesis in the literature.

2.7.1 Overview

The aim of this thesis is to solve a mapping problem with several requirements. First, heuristics must be scalable. It is also required that the approach should be able to easily map as much as tens and millions of tasks in a reasonable amount of time while maintaining good solution quality. Second, the optimization constraint which must be considered are capacity constraints. The main will is to maximize the occupation of processors while minimizing communication costs. In ΣC, introduced in Section 1.6, constraints depends on the CPU and the memories in embedded systems. If these constraints are met, the application works fine and each execution cycle is performed in bounded time. We can cite image processing operations as an example. Sixty images must be displayed each second. One way to have such a fast execution consists in placing the adequate task on the suitable processor in order to reduce execution time.

Table 2.1 and Table 2.2 present an overview filtered using the 3 just defined requirements :

- scalability,
- topology-awareness of the target architecture,
- optimization constraints.

Clearly, all exact methods of the literature are eliminated because, despite of their ability to find the optimal solution, the problem size prevents execution in reasonable time. They could be used to compare the solution quality of our heuristic for small instances, but it is not sure that the heuristic behavior will remain the same on 100 tasks than on $250,000$ tasks.

Considering the one-phase mapping heuristics: only few articles present an approach able to solve the mapping problem with capacity constraints and topology awareness of the target. However, they are not able to scale on instances which present more than $2,000$ tasks. Some greedy or iterative approaches are able to deal with important numbers of vertices but either they do not meet the requirements, or solution quality is relatively poor.

Concerning the two-phases mapping heuristics, spectral bisection may provide very good quality results but they face the same scalability problem. For partitioning algorithms, the graph partitioning problem is NP-hard, which means while the number of tasks is greater than a certain limit, the complexity of finding a good partitioning becomes equivalent to the complexity of finding a good mapping. Moreover, capacity constraints add some complexities. For clustering heuristics, it is possible to transform the obtained clusters and to map them onto processors. However, the application size remains a limit which has not been overcome yet.

This led us to multilevel approaches. Many heuristics are able to deal with a high number of tasks and, above all, are topology aware. In contrast to all just cited heuristics, no multilevel heuristic deals with capacity constraints because it uses heuristics which do not deal with this type of constraint. Most of those heuristics focus on the load balancing problem.

2.7.2 Our work

The mapping heuristics which are developed in this thesis are focused on dataflow process networks. These heuristics are scalable, topology-aware of the target architecture and are dependent of capacity constraints.

One of the three heuristics is a two-phases mapping. First, a cluster of tasks is built using graph exploration algorithm. Second, the cluster is either merged to another cluster or directly mapped onto the corresponding processor.

The two others are one-phase mapping heuristics. One consists in a greedy heuristic which maps tasks one after another using an affinity notion which is explained in Chapter 3; it also uses graph exploration algorithms. The second one, which is explained in Chapter 4, uses game theory principles in order to select how to map tasks.

2.8 Conclusion

In this chapter, the importance of the mapping problem has been emphasized. The state of the art has been organized using the number of mapping phases which exists in approaches used to solve the mapping problem. For one-phase methods, mapping approaches, exact methods, evolutionary algorithms, and greedy or iterative heuristics have been highlighted. Concerning two-phases methods, methods have been split in the following categories: partitioning, clustering, multilevel and spectral bisection. For each element of the literature, a brief description of the mapping methods or partitioning and mapping approach has been provided. Then, the requirements our problem is depending on, have been highlighted from all works, that is to say: scalability, topology awareness and capacity constraints. At the end, it has been noticed that many approaches combined two of these three requirements but never met all of them. This evidence of this lack in the literature motivated us in the development of the three heuristics which will be presented in the following chapters.

Name	Constraints	# of tasks	Topology aware
(PT-)SCOTCH ([142]) [145] Pellegrini *et al.*	Load Balancing	$\leq 10^9$	Yes
(Par)Metis ([100]) [95] Karypis, Kumar *et al.*	Load Balancing	$\leq 10^8$	No
ΣC toolchain - Aubry *et al.* Partitioning & Placing [174] Sirdey *et al.*	Knapsack	< 2000	Yes
ΣC toolchain - Aubry *et al.* Parallel Simulated Annealing [65] Galea *et al.*	Knapsack	< 2000	Yes
Energy and Perf. Aware Mapping for Regular NoC Architecture [87] Hu *et al.*	Bandwidth	≤ 50	Yes
Multiobjective Optimization and evolutionary algorithms for the application mapping problem in MPSoC [45] Erbas *et al.*	Knapsack	≤ 1000	Yes
Automated memory-aware application distribution for MPSoC [139] Orsila *et al.*	Communication to computation ratio	≤ 100	No
Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints [127] Manolache *et al.*	Deadline miss ratio	≤ 40	No
Communication-aware allocation and scheduling framework for stream oriented MPSoC [156] Ruggiero *et al.*	Knapsack	≤ 10	No
An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms [22] Bonfietti *et al.*	Load Balancing	≤ 250	Yes
Efficient resource mapping framework over networked clouds via iterated local search based request partitioning [116] Leivadeas *et al.*	Bandwidth	≤ 350	No

Table 2.1: Related works

Name	Constraints	# of tasks	Topology aware
Cluster partitioning approaches to mapping parallel programs onto a hypercube [157] Sadayappan *et al.*	Load balancing	≤ 1449	Yes
Topology aware task mapping for reducing communication contention on large parallel machines [2] Agarwal *et al.*	Load balancing	≤ 3240	Yes
Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems [12] Barnard *et al.*	Load balancing	$< 10^5$	No
Multilevel reactive tabu search for graph partitioning [15] Battiti *et al.*	Load balancing	≤ 65536	No
Graph bisection algorithm with good average case behavior [25] Bui *et al.*	Load balancing	≤ 800	No
Jostle [192] Walshaw *et al.*	Load balancing	$\leq 10^7$	Yes

Table 2.2: Related works

Chapter 3

Two Scalable Mapping Methods for Unitary Weighted Task Graphs

Contents

3.1 Introduction

In this chapter, the aim is to develop scalable, topology-aware of the target architecture and constraints-based mapping methods. The task mapping problem can be interpreted as a graph mapping problem. The application is modeled as an undirected graph where vertices are tasks and an edge between two tasks exists if and only if the edge is valued by a non-zero bandwidth. We wanted to start our research by considering unitary weighted tasks in order to facilitate the problem and to determine approaches that may be used on non-unitary weighted tasks. This first step consists in establishing such a method where all task or communication channel weight values are set to one. The main idea consists in taking advantage of the task graph topology in order to gather on the same processor tasks which have the lowest distance among each other. Once terminologies of graph theory have been introduced, two static mapping heuristics are presented. First, a two phase-mapping heuristic, denoted as Subgraph-Wise Placement (SWP), is introduced. This heuristic is an iterative process in which, at each step, a subset of unassigned tasks is determined using a breadth-first search strategy. The obtained subgraph is then mapped onto the node which satisfies the most a notion of affinity, among nodes of sufficient available capacity for all tasks of the subgraph. A second heuristic is then presented.

It is a one phase mapping heuristic referred to as Task-Wise Placement (TWP). In this heuristic, all tasks are mapped one after another using a notion of task-to-node affinity.

The two heuristics are then evaluated in a comparative study with a method which has been presented in the state of the art [174] and is currently used in the ΣC toolchain; they are also compared to random based mappings, as discussed in Section 1.8.

3.2 A Two Phase Mapping Method: Subgraph-Wise Placement (SWP)

In this section, a two phase mapping method, denoted as Subgraph task-wise placement, is presented. A subset of tasks referred to as subgraph is built using an adapted version of the BFT presented in Subsection 3.2.1. Once the subgraph is built, a metric called affinity is computed in order to evaluate the subgraph. The metric is detailed in Subsection 1.9.3. If the node with the greatest affinity towards the subgraph has enough space left, tasks which compose the subgraph are mapped onto the node. The pseudo-code is detailed by Algorithm 3.

3.2.1 Creation of a Subgraph of Tasks

Working with applications that are modeled by dataflow process networks implies that these types of graphs are connected. The main idea is to take advantage of this feature by gathering tasks which are communicating together and building a connected subgraph. As explained in Subsection 1.9.2, breadth-first traversal is able to provide a task ordering where all tasks in the assembled subsets are near to each other. However, it is not possible to map all the task graph onto one processor because of the node capacity limits. This means that the sum of weights of all tasks must not exceed this capacity. The size of the subgraph corresponds to the sum of all weights of its members. This size must be limited in order to respect the capacity constraint. Moreover, another feature has to be considered. While assigning tasks to processors, the remaining capacity of each node where subgraphs are mapped is decreasing. Due to the fact that remaining capacities on nodes may vary, the size of the subset is computed by determining the maximal remaining node capacity and this value is multiplied by a factor $\frac{1}{2}$ as shown in the following equation:

$$size = \frac{\max\limits_{\forall n \in N} RC_n}{2} \ , \tag{3.1}$$

where RC corresponds to an array containing the remaining capacities of each node. The factor $\frac{1}{2}$ has been added because empirical results show that best performance is obtained using this factor.

The starting task selection may be one of the most important operation of the whole computation. Indeed, a bad task selection leads to a very bad mapping because it favors the emergence of singletons, that is to say, a task whose all neighbors are already mapped onto a node. Unfortunately, it may happen that these singletons are mapped onto nodes

which are located far away from the nodes to which the neighboring tasks of the singletons are mapped. This aspect drastically degrades the global mapping quality. This is one reason why this step plays a major role. In order to avoid singletons, the task with the lowest number of neighbors is determined and used as starting task for the breadth first traversal algorithm. Tasks with the lowest number of neighbors are often located at the extremity of a graph. By starting with these tasks, singleton are reduced and we reached a high number of connected tasks.

Now that the starting task determination strategy and the size of the subgraph to compute have been explained, the BFT will be computed. However, it has to be adapted to take into account different features like task weights and the size of the subgraph to generate. Algorithm 1 describes the process. It is a classic BFT process with an additional stop criterion which is the sum of weights of all the already explored tasks. When this sum reaches the expected size, the execution of the algorithm stops.

Algorithm 1 Breadth-first traversal for SWP (BFT1)

Input: $G = (T, E)$ ▷ Task graph.
Input: t_0 ▷ Starting task.
Input: $size$ ▷ Maximum threshold of the subgraph.
Input: $W^{|T|}$ ▷ An array which contains the weights of all tasks.
Input: $M^{|T|}$ ▷ An array indicating if a task is flagged on not. $M[v] = 1$ if v explored, 0 otherwise.
Output: S ▷ Subgraph to build.
Output: $M^{|T|}$ ▷ Updated with the new flagged tasks.
Output: $sgWeight$ ▷ The sum of weights of all tasks in the subgraph.

```
Q: ▷ A FIFO queue.
pos ← 0
inc ← 1 ▷ Number of tasks in the subgraph.
sgWeight ← 0
Q ← {t_0}
sgWeight ← W[t_0]
while sgWeight ≤ size and pos < inc do
  | t ← Q.pop
  | if M[t] = 0 and sgWeight + W[t] ≤ size then
  |   | inc ← inc + 1
  |   | S ← v
  |   | M[t] ← 1
  |   | sgWeight ← sgWeight + W[t]
  |   | for all t' ∈ N_G(t) and M[t'] = 0 do
  |   |   | Q.push(t')
  | pos ← pos + 1
return S, M^{|T|}, sgWeight
```

Now that the rules which allows to determine a subgraph out of a task graph has

been identified, the attention is focused on how to estimate on which node the subgraph should be mapped onto.

3.2.2 Subgraph to node affinity

The aim is to determine which node is the most suitable to contain the subgraph. Clearly, the remaining capacity must be greater than the size of the subgraph. The idea is to gather as many tasks as possible which have the strongest edge weights on the same node. One way to determine where the subgraph has to be mapped onto, it is virtually mapped onto one node after another. For each node, the subgraph to node affinity is computed using the affinity notion detailed in Subsection 1.9.3. The affinity equation 1.10 is adapted for this computation :

$$Aff_{SN} = \sum_{\forall t \in S} \sum_{\substack{\forall t' \in t \\ t \neq t'}} q_{tt'} \ . \tag{3.2}$$

Once the computation is over, the highest measured affinity indicates the most suitable node the subgraph should be mapped onto. Algorithm 2 describes the affinity computation.

Algorithm 2 Computation of subgraph to nodes affinity.

Input: $G = (T, E)$ ▷ task graph with q bandwidth matrix.
Input: $G' = (N, E')$ ▷ node graph with d distance matrix.
Input: $S \in T$ ▷ the subgraph computed by BFT1 algorithm.
Input: $S2NA^{|N|}$ ▷ Task to Nodes affinity array.
Input: $sgWeight$ ▷ Sum of all weights of tasks in the subgraph.
Input: $RC^{|N|} \in \mathbb{N}^+$ ▷ Node remaining capacity array.
Output: $node$ ▷ the node on which the subgraph should be mapped.

```
maxAff ← 0                                   ▷ the maximal affinity value.
A^|T|                                        ▷ An array containing the node where t' is mapped.
for all t ∈ S do
    for all t' ∈ N_G(t) do
        node ← A[t']
        if node ≠ ∅ then
            S2NA[node] ← S2NA[node] + val[t']
node ← argmax_{n∈N, RC[n]<sgWeight} (S2NA[n])
return node
```

3.2.3 Complexity of the algorithm

Computing the affinities of one subgraph to all nodes as presented in Algorithm 4 has an average complexity of $\mathcal{O}(\frac{|E|}{|N|})$, since $\frac{|E|}{|T|}$ is the average degree δ of G which is computed

line 4. The affinity is computed for each task in the subgraph line 3 in $\frac{|T|}{|N|}$, which is the maximal number of tasks in one subgraph.

For the following complexity computation, let X be the number generated subgraph. The focus is now laid on the SWP algorithm detailed in Algorithm 3. It is important to note that the "while" loop of line 11 depends on the number of created subgraphs. This means, this loop is computed X times.

Inside the loop, the maximum complexity of all statements is $\mathcal{O}(|E|+\frac{|E|}{|X|}\times\delta)$. Indeed:

- At line 12, the search for the minimum edge value is performed in $\mathcal{O}(|E|)$.
- At line 16, the affinity computation depends on the size of the subgraph. The worst case consists in computing the BFS on the whole task graph. Thus, the worst complexity of this computation is $\mathcal{O}(|E|)$.
- At line 17, the number of tasks depends on the number of tasks in the current subgraph, that is $\frac{|E|}{|X|}$.
- At line 21, the number of tasks corresponds to the average degree of G which is $\frac{|E|}{|T|}$.

The worst case for the number of subgraphs is the number of nodes times the node capacities C. C is a constant value which is the same for all nodes in this context. This means that $X = |N| \times C$.

Therefore, the overall complexity of SWP is $\mathcal{O}(|T| + C|N||E| + \frac{|E|^2}{|T|})$. Because this complexity depends on the node capacity which is given as an input, the complexity of SWP is pseudo-polynomial.

3.2.4 Conclusion

In this section, a two phase mapping heuristic denoted as subgraph-wize placement is presented. The first phase constructs a subset of tasks using an adapted breadth-first traversal algorithm. The second mapping phase is performed by mapping the subset of tasks on the affinity-based determined node. Experimental results are presented in section 3.4.

Now, another mapping heuristic will be presented. Instead of processing the mapping in two phases, this alternative consists in determining a one-phase mapping heuristic.

3.3 One Phase Mapping Method: Task-Wise Placement (TWP)

In this section, a one-phase mapping heuristic, denoted as Task-Wise Placement (TWP), is presented. Instead of partitioning a set of tasks and mapping it as shown in SWP, tasks are rather mapped one after another on nodes which are given preference according to an affinity function specially developed for that purpose. Despite of the fact that the

Algorithm 3 Subgraph Wise Placement(SWP)

Input: $G = (T, E)$: task graph with q bandwidth matrix
Input: $G' = (N, E')$: node graph with d distance matrix
Input: $RC^{|N|} \in \mathbb{N}^+$: node remaining capacity array
Input: $W^{|T|} \in \mathbb{N}^+$: task weight array
Output: A: Task mapping assignment array of size $|T|$

$S2NA \leftarrow \{0\}^{|S|\times|N|}$ ▷ Subgraph to nodes affinity matrix
$M \leftarrow \{0\}^{|T|}$ ▷ Tasks marking array
$RC \leftarrow \{0\}^{|N|}$ ▷ Node Remaining Capacity
$count \leftarrow 0$ ▷ An assigned task counter
$n_1 \leftarrow 0, n_2 \leftarrow 0, n \leftarrow 0$ ▷ nodes identifier
$S2NAff \leftarrow 0$ ▷ Maximal value of $S2NA$
$sgWeight \leftarrow 0$ ▷ Sum of all tasks weight in S
$degrees \leftarrow \{0\}^{|T|}$ ▷ array array of degrees of each task
for all $t \in T$ **do**
  $degrees[t] \leftarrow |N_G(t)|$
while $count < |T|$ **do**
  $t \leftarrow \underset{\substack{t\in T \\ A[t]\neq\varnothing}}{\operatorname{argmin}}\left(\sum_{t'\in N_G(t)} q_{tt'}\right)$
  $size \leftarrow (\max_{n\in N} RC[n]) \times 0.5)$
  $S \leftarrow \{0\}^{size}$
  $(S, sgWeight) \leftarrow BFT1(G, t, size, M, W)$
  $n \leftarrow \text{computeS2N}(S, G, G', S2NA, RC, sgWeight)$
  for all $t \in S$ **do**
    $A[t] \leftarrow n$
    $M[t] \leftarrow 1$
    $RC[n] \leftarrow RC[n] - W[t]$
    for all $t' \in N_G(t)$ **do**
      if $M[t'] = 0$ **then**
        $degrees[t'] \leftarrow degrees[t'] - 1$
  $count \leftarrow count + 1$
return A

mapping is done in one phase, the mapping procedure is split in two sub-procedures: the mapping procedure and the node saturation algorithm.

First, all tasks are mapped using a notion of distance affinity which will be introduced in Subsection 3.3.1. This operation lasts until one node gets saturated. When this occurs, all tasks, with their greatest affinity toward the saturated node, are reinitialized in the affinity matrix. The saturated node is removed of the candidate nodes set. Tasks, whose affinity towards this node is maximal are also removed from the waiting set which contains all marked tasks to map.

Before giving further details, it is necessary to define a metric which indicates on which node each task must be mapped. The affinity notion detailed in Subsection 1.9.3 computes affinity for sets. Therefore it is necessary to adapt it so that it only computes task to node affinities.

3.3.1 Distance affinity

When iteratively choosing on which node a task must be placed, an intuitive way is to place it as close as possible from its neighboring tasks which are already placed. This leads to the determination of the following equation. A notion of distance affinity is introduced. For any DPN mapping solution, the distance affinity β_{tn} between a task t and a node n is represented by the following equation:

$$\beta_{tn} = \sum_{n' \in N} \sum_{t' \in T} x_{t'n'} x_{tn} q_{tt'} \times \frac{1}{2 \times d_{nn'} + 1} \ , \tag{3.3}$$

where :

- $G = (T, E)$ is the task graph to map,
- $G' = (N, E')$ is the node graph of the target architecture,
- $x_{tn}, x_{t'n'}$ are boolean values indicating if task t and t' are respectively mapped onto node n and n'
- $d_{nn'}$ is the distance between node n and n', and
- and $q_{tt'}$ is the bandwidth between task t and task t'.

The equation can be described by the following: an available node is defined by a node which has enough remaining space in order to host the task. The sum of each independent contribution of each task in the neighborhood of t is computed. The equation is proportional to the bandwidth $q_{tt'}$ between all tasks processed during this summation. When the distance between two nodes increases, the distance affinity decreases. This equation favors the mapping of tasks that communicates the most with t.

The final summation of the obtained results for each task corresponds to the affinity of task t to node n.

3.3.2 The mapping procedure

In this method, all tasks are assigned one after another using the distance affinity as the criterion for choosing the next task to be placed.

Initially, the algorithm first determines the task with the greatest sum of the bandwidths of adjacency edges. All of its neighbors are inserted in a candidate set. All tasks presents in this candidate set are considered as candidate tasks. For each candidate task, the distance affinity is computed and this computation allows to determine the more suitable node onto which the candidate task should be mapped. Once all distance affinities are computed, the (task,node) pair with the highest affinity is selected. If two or more pairs have the same affinity, the priority corresponds to the FIFO order. The selected task is then placed in the corresponding node.

This operation lasts until it is no longer possible to map a task to its corresponding node because the node has not enough space left for the task.

After a while, it is obvious that nodes get saturated, so that no tasks can no longer fit in the node. However, many candidate tasks are still expected to be mapped onto it because of their affinities. The corresponding obsolete pairs must be therefore removed from the candidate set. All nodes, which are not saturated, in the neighborhood of the saturated node are used as a new basis for the mapping. In addition, a pre-generated ordering of all tasks, generated by breadth first traversal, performed at the beginning of the algorithm, is processed for choosing as many unassigned tasks as nodes which have been selected. Basically, each of the first unassigned tasks in the ordering are assigned to a different selected node. The unassigned neighbors of those tasks are then placed in the queue and their respective affinities are updated.

The whole process repeats itself as long as unassigned tasks remain.

Algorithm 4 updateAffinities algorithm computing the distance affinities for the input task toward all nodes. The complexity of the algorithm is $\mathcal{O}(tn)$

Input: $G = (T, E)$: task graph with q bandwidth matrix
Input: $G' = (N, E')$: node graph with d distance matrix
Input: A: Task mapping assignment array of size $|T|$
Input: t: task whose affinities have to be computed.
Output: $T2NA$

1: $T2NA^{|N|} \leftarrow \{0\}^{|N|}$: Task to Nodes affinity array of t to all nodes.
2: **for all** $t' \in N_G(t)$ **do**
3: | $node \leftarrow A[t']$
4: | **if** $node \neq \varnothing$) **then**
5: | | **for all** $n \in N$ **do**
6: | | | $T2NA[n] \leftarrow T2NA[n] + q_{tt'} \times \frac{1}{2 \times d[node][n]+1}$
7: **return** $T2NA$

Algorithm 5 Task-Wise Placement (TWP)

Input: $G = (T, E)$: task graph with q bandwidth matrix
Input: $G' = (N, E')$: node graph
Input: Q_BFT: BFT algorithm result starting from t_0
Input: $C \in \mathbb{R}$: node capacity
Output: A: Task mapping assignment array of size $|T|$

$T2NA \leftarrow \{0\}^{|T|\times|N|}$ ▷ Tasks to nodes affinity matrix
$t_0 \leftarrow \operatorname{argmax}_{t\in T}(\sum_{t'\in N_G(t)} q_{tt'})$ ▷ Starting task
$Q \leftarrow N_G(t_0)$ ▷ Candidate set for tasks to be placed next
$count \leftarrow 0$ ▷ A assigned Task counter
$A \leftarrow \{\varnothing\}^{|T|}$ ▷ All vertices are initially unassigned
$A[t_0] \leftarrow 0$ ▷ Assign t_0 to arbitrary node
for all $t' \in N_G(t_0)$ **do**
  $T2NA[t_0] \leftarrow \text{updateAffinities}(G, G', A, t')$
$count \leftarrow count + 1$
while $count < |T|$ **do**
  if $count > 1$ **then**
    $t_1 \leftarrow \operatorname{argmax}_{t\in T}(\sum_{t'\in N_G(t)} q_{tt'})$
    $Q \leftarrow \{t_1\}$
    for all $t' \in N_G(t_1)$ **do**
      $T2NA[t_1] \leftarrow \text{updateAffinities}(G, G', A, t')$
    $count \leftarrow count + 1$
  while $|Q| > 0$ **do**
    $(t, n) \leftarrow \operatorname{argmax}_{(t',n')\in Q\times N} (T2NA[t'][n'])$
    $Q \leftarrow Q \backslash \{t\}$
    if $w_t + \sum_{t':A[t']=n} w_{t'} \leq C$ **then** ▷ n is not saturated
      $Q \leftarrow Q \cup N_G(t)$
      $A[t] \leftarrow n$
      $count \leftarrow count + 1$
      for all $t' \in N_G(t)$ **do**
        $T2NA[t] \leftarrow \text{updateAffinities}(G, G', A, t')$
    else
      $Q \leftarrow Q \backslash \{t \in Q : \operatorname{argmax}_{n'\in N}(T2NA[t][n']) = n\}$
      for all $n' \in N_{G'}(n)$ **do**
        if n' not saturated **then**
          repeat
            $t \leftarrow \text{deQueue}(Q\_BFT)$
          until $A[t] = \varnothing$
          $Q \leftarrow Q \cup N_G(t)$
          $A[t] \leftarrow n$
          $count \leftarrow count + 1$
          for all $t' \in N_G(t)$ **do**
            $T2NA[t] \leftarrow \text{updateAffinities}(G, G', A, t')$
return A

3.3.3 Complexity of the Algorithm

TWP is presented in Algorithm 5. The whole algorithm will be detailed and the algorithm complexity is also analyzed.

Theorem 2. *The complexity of TWP is* $\mathcal{O}(|T|^2|N|)$.

Proof. Computing the distance affinities of one task to all nodes as presented in Algorithm 4 has an average complexity of $\mathcal{O}(\frac{|N||E|}{|T|})$, since $\frac{|E|}{|T|}$ is the average degree of G.

It is important to note that both nested "while" loops starting from line 10 and 17 actually span all tasks one by one, so the overall complexity of both loops, without considering their content, is $\mathcal{O}(|T|)$.

Inside both loops, the maximum complexity of all statements is $\mathcal{O}(|T||N|)$. Indeed:

- At line 11, the search for the best (task,node) pair is performed in $\mathcal{O}(|Q||N|)$ where Q is the size of the candidate set. We have no precise estimation for the size of Q since it strongly depends on the topology of G, and the order tasks are processed. We can only affirm that $|Q| \leq |T|$, hence a complexity of $\mathcal{O}(|T||N|)$.

- At line 20, the test for node saturation is actually implemented using an array of remaining capacities, in nodes, updated in $\mathcal{O}(1)$ when each task is placed. So the test is bounded by $\mathcal{O}(1)$.

- At line 27, the search for tasks to be removed from the candidate set is $\mathcal{O}(|Q||N|)$, which is bounded by $\mathcal{O}(|T||N|)$.

- Different calls to Algorithm 4 are done in $\mathcal{O}(\frac{|N||E|}{|T|})$. When called for all neighbors of a task, the overall complexity is multiplied by the average degree hence a complexity of $\mathcal{O}(\frac{|N||E|^2}{|T|}^2)$. Since $\mathcal{O}(\frac{|E|^2}{|T|^2}) < |T|$ for large enough sizes of T, the complexity is bounded by $\mathcal{O}(|N||T|)$.

Therefore, the overall complexity of TWP is $\mathcal{O}(|T|^2|N|)$. □

3.3.4 Conclusion

Now that the one phase mapping heuristic has been defined, performance of both mapping methods will be analyzed. In the next section, experiments on several types of instances will be performed in order to compare the performance of our algorithms.

3.4 Results

3.4.1 Execution Platform

The target system is a PC based on the Intel Xeon E5/Core i7 processor running at 2.0 GHz. As our algorithms are purely sequential, only one CPU core is used.

Name	# tasks	# nodes	node capacity
grid12x12	144	4	40
grid23x23	529	16	40
grid46x46	2,116	64	40
grid100x100	10,000	256	40

Table 3.1: Grid shaped task topologies.

Name	# tasks	# nodes	node capacity
b12	1,065	36	40
b17	24,171	256	100
b18	114,561	400	300
b19	231,266	576	410

Table 3.2: Logic gate network topologies.

3.4.2 Instances

Two kinds of task graph topologies were used. First, a set of grid-shaped task topologies, which correspond to dataflow computational networks like matrix products (Table 3.1). The other kind of task graph is generated out of logic gate networks resulting in the design of microprocessors. These configurations of task networks typically can be found in real life complex dataflow applications (Table 3.2).

The node layout is a square torus, hence the number of nodes in all instances is a square value.

For each pair (t, t') of tasks of the grid, the bandwidth $q_{tt'}$ is set to 1 if tasks t and t' are adjacent in the task grid, and 0 otherwise. For graphs generated out of logic gate networks, the edge weights are the number of arcs between the corresponding elements in the original multigraph.

For each pair of nodes (n, n'), the distance $d_{nn'}$ is the Manhattan distance between nodes n and n'.

In those experimentations, all instances are limited to one resource and the resource occupation of every task in arbitrarily set to 1.

3.4.3 Computational results

We compare our methods with that of [174] which is a two-phase method where the tasks are first partitioned into as many partitions as there are processors, then the partitions are mapped onto the processors. We denote this method as Partition and Place (P&P). It is the only other method we know to solve the DPN mapping problem.

All tables display the solution objective value and the execution time of the methods we cited. The tables display the application of the algorithms on grid-shaped task topologies and on logic gate network topologies (Table 3.3). The results are also illustrated by Figure 3.1 and Figure 3.2. We can observe that, for small instances of grids, the P&P algorithm provides better results than TWP algorithm whereas that TWP is faster.

Name	P&P		SWP		TWP	
	Sol. Val.	run time	Sol. Val.	run time	Sol. Val.	run time
grid12x12	37	0.02 s	75	84 μs	41	2.3 ms
grid23x23	220	0.05 s	338	400 μs	338	5.2 ms
grid46x46	2,500	2 s	2,565	9.93 ms	2,306	0.17 s
grid100x100	45,613	240 s	18,700	0.49 s	16,000	3 s
b12	1,200	4.85 s	2,205	5.77 ms	1,598	9.6 ms
b17	135,000	3,100 s	155,396	2.71 s	109,879	88 s
b18	832,538	40 h 18 min	1,936,952	22.25 s	395,624	2,163 s
b19	-	-	5,613,634	75.24 s	-	-

Table 3.3: P&P , TWP and SWP approach on grids and LGNs.

Name	P&P	SWP	TWP
	Ratio	Ratio	Ratio
grid12x12	7.14	3.52	6.43
grid23x23	9.2	5.98	5.99
grid46x46	6.62	6.45	7.18
grid100x100	3.47	8.47	9.9
b12	5.57	3.04	4.19
b17	3.13	2.72	3.85
b18	10.25	4.40	21.58
b19	-	3.63	-

Table 3.4: $\frac{Random}{P\&P}$, $\frac{Random}{TWP}$, $\frac{Random}{SWP}$ ratios on grids and LGNs.

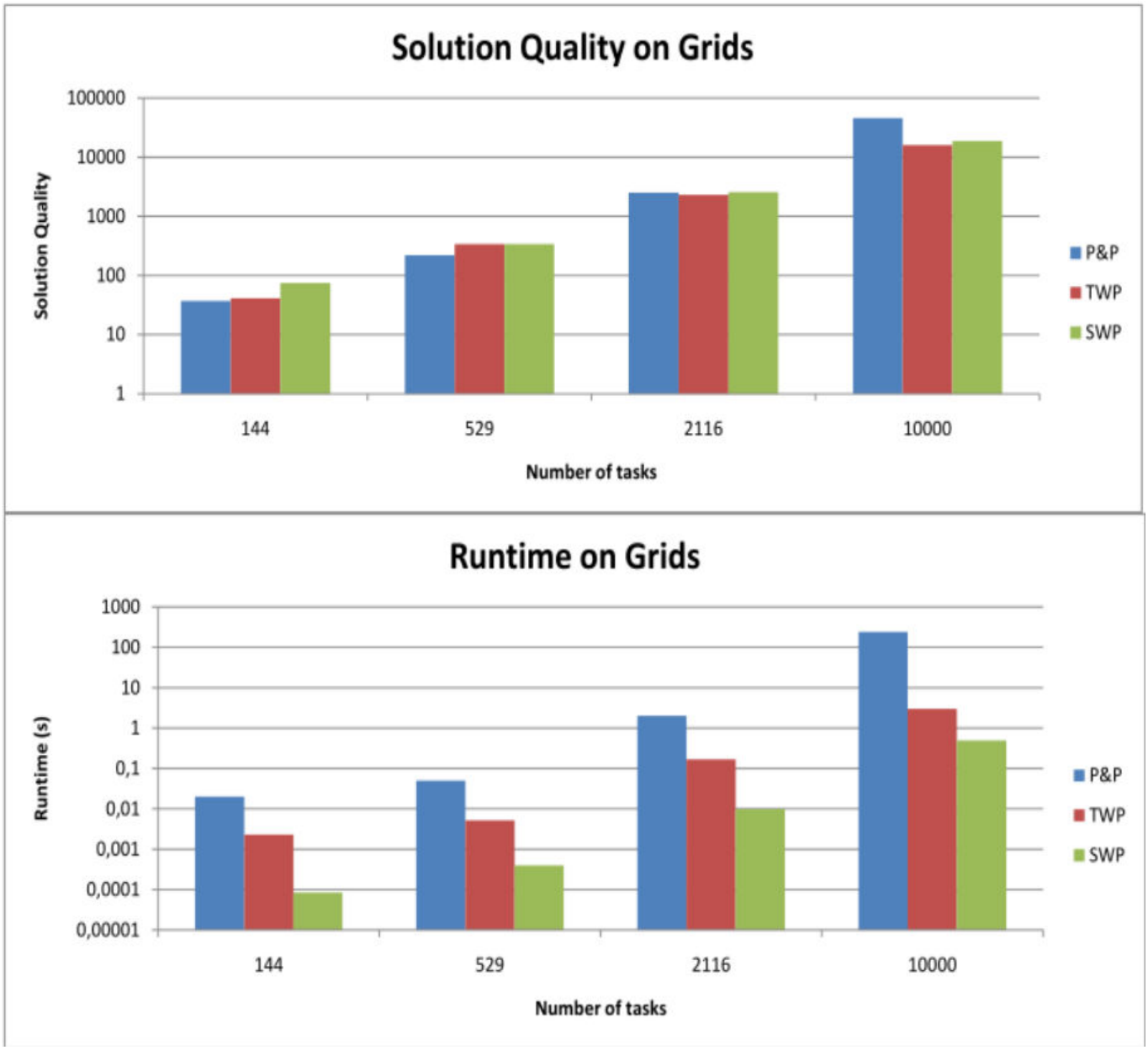


Figure 3.1: Solution qualities and execution times of P&P, TWP and SWP on grids.

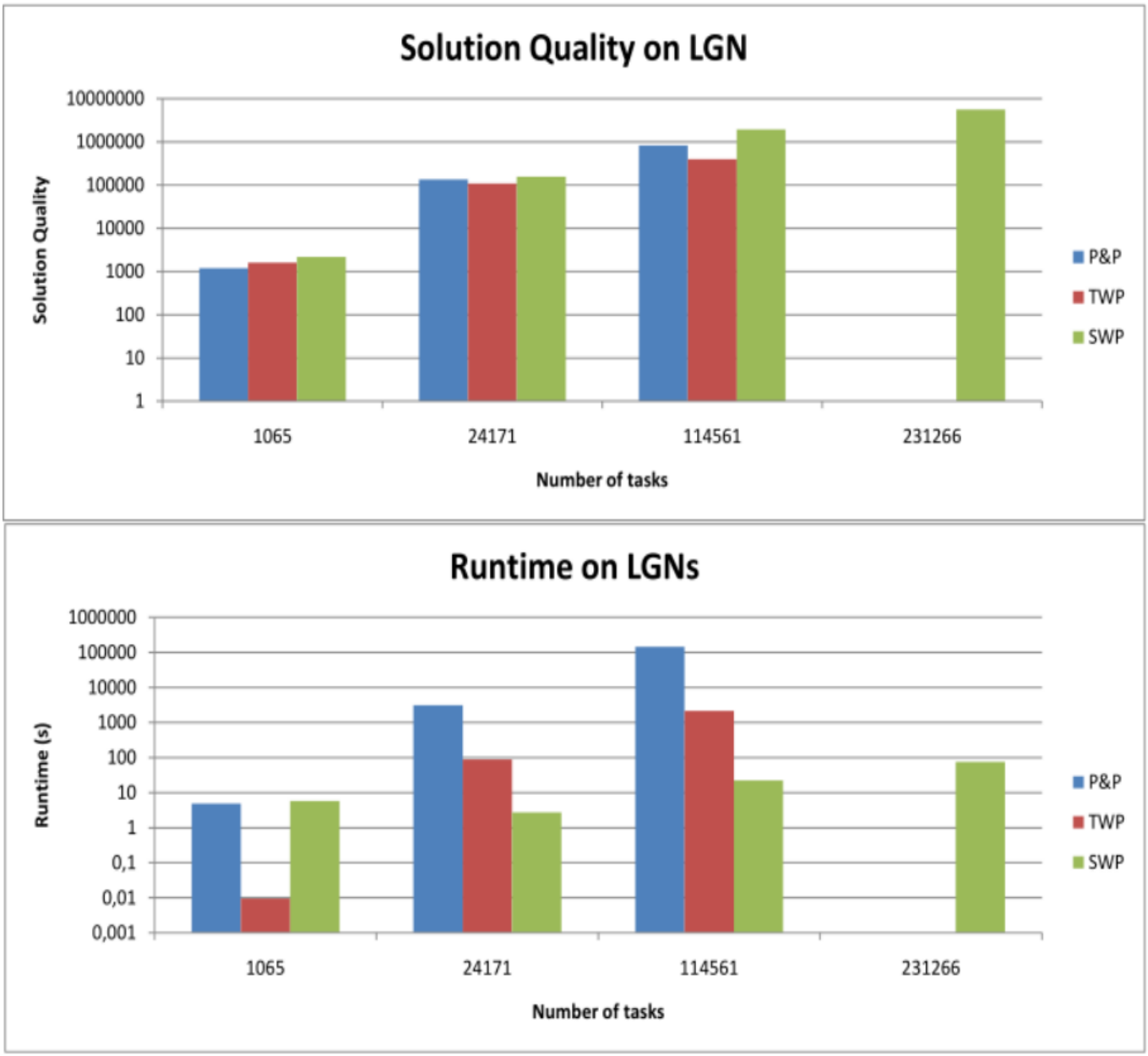


Figure 3.2: Solution qualities and execution times of P&P, TWP and SWP on grids.

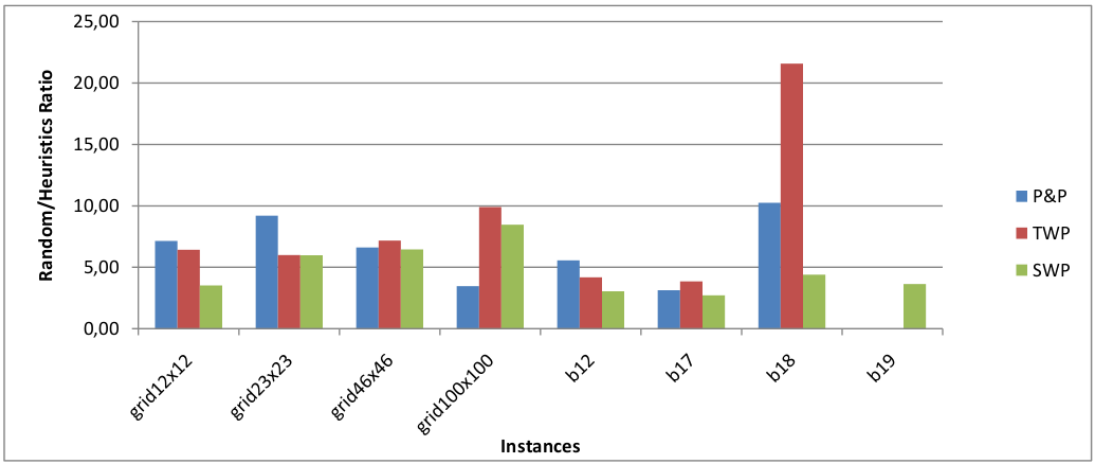


Figure 3.3: $\frac{Random}{P\&P}$, $\frac{Random}{TWP}$, $\frac{Random}{SWP}$ ratios on Grids and LGNs.

However when the number of tasks is higher than 2000, TWP begins to provide better results and far better execution times. The solution quality of TWP tends to get better than P&P when the number of tasks increases, with a relative speedup of 67 for the b18 instance. One fact that calls our attention is the fact that the TWP algorithm works fine either on logic gate network topologies or grid-shaped task topologies. However, the algorithm cannot give any results in a reasonable amount of time for instances of $200,000$ tasks.

The focus is now set on SWP. Execution times are several orders of magnitude faster than the P&P approach, while providing solutions whose quality tends to get comparatively similar or better on the largest instances. By comparing the difference between SWP and TWP, the TWP method provides better results than the subgraph method, while the subgraph method runs faster than the TWP method and scales easily on very large instances.

Now, the interest is set on the ratios obtained by comparing results obtained by the random based mapping metric defined in Section 1.8 (Table 3.4). Results are displayed in Figure 3.3. For all instances, heuristics are at least 2 to 21 times better than random based mapping. One can observe that for small instances of either grids or LGN, P&P provides better ratios than SWP and TWP. Yet on larger instances, this tends is reversed. This is not surprising and has already be pointed out above. Moreover, by having a deeper look, we can notice that when the number of tasks increases, the $\frac{Random}{P\&P}$ ratio decreases. However, for SWP and TWP, on grids either, while the number of tasks increases, the ratio also increases. This confirms that both algorithms have a better behavior on large instances. On LGN instances, it seems that this type of graph is harder to map for all three heuristics. Even if TWP outperforms P&P which outperforms SWP as mentioned above, heuristics behavior cannot be interpreted by only making conclusions on the number of tasks. Moreover, LGN topologies are not likely DPN topologies which can be represented as series-parallel graphs. Grids are similar to this type of graphs and the fact that ratios are higher on large graphs shows that TWP and SWP may be efficient on it. The metric

was able to show the difference of efficiency of the heuristics between the different graphs topologies which have been used.

The increase in terms of compared solution quality between our methods and the P&P algorithm finds its explanation in two different aspects. First, as the partitioning phase of P&P does not take node distance into account, tasks are gathered together with no knowledge of the destination processor topology. Thus, choices made during this phase may undermine the overall solution quality. In the opposite, the distance affinity notion we use in the TWP approach allows us to take profit of the topology and avoid many bad choices. Second, even not taking profit from the node distances, the subgraph placement method has the advantage that it tries to avoid placing singletons or very small subgraphs, while the last 10% (or perhaps more) of the tasks to be assigned in P&P may probably not be efficiently assigned, leading to a drop in quality.

3.5 Conclusion

In this chapter, two mapping heuristics have been developed. Subgraph-Wise Placement heuristic is a two-phase mapping. First a subgraph is built using the breadth-first traversal algorithm with several parameters. Second the mapping phase occurs by mapping the subset of tasks on the corresponding node using a notion of affinity. On the other hand, the Task-Wise Placement heuristic is a one-phase mapping which assigns each task one after another using a distance affinity metric. The whole set of tasks is ordered using BFS and used when the need to populate an empty node appears.

These two approaches are scalable, topology-aware and work with knapsack constraints. Obtained results are satisfying on unitary-weighted tasks graphs. Now, the focus will set on non-unitary weighted task graphs. For these types of task graphs, using locality of tasks is not enough, some characteristics have to be added in the algorithmic approach. This leads us to the next chapter which present a new mapping heuristic based on regret theory.

Chapter 4

A Regret Theory-Based Mapping Method for Non-Unitary Weighted Task Graphs

Contents

4.1 Introduction

In this chapter, the goal consists in setting up a mapping method able to deal with homogeneous target architectures, but with non-unitary weighted task graphs. The difficulty lies in the way tasks are mapped because the mapping presents a high uncertainty level. It is not known if an intermediate partial solution at time t may lead to final disastrous global mapping quality at the end of the mapping procedure. One approach able to deal with this concept of quality fluctuation which depends on the system evolution is Game Theory. This mathematical field originates from economic science and is a powerful tool which contains methods that are effective in case of uncertainty like the model we are working on. More precisely, the focus will be set on a sub-domain of game theory: repeated behavioral games. This sub-domain features a portfolio of algorithms and decision making techniques. One of these techniques is denoted as "regret theory". In this work, regret theory is applied in the task selection process in order to allow the development of

a new mapping algorithm able to map tasks while paying attention to the impact of not mapping a task at any solution construction step.

In order to verify the strength of the algorithm, it is tested on several types of task graph instances. For all types of task graph, a pool of different task and edge weighted instances are generated. At the time of writing of this dissertation, no comparable heuristic could be found. The metric defined in 1.8 will help in the evaluation of the results. In addition, an adapted version of TWP is used as comparative heuristic. Once performance of the regret heuristic have been detailed, a GRASP procedure is set up in order to improve the solution quality of the presented heuristic and also determines how many runs have to be performed in order to get the best generated solution quality.

4.2 Game theory background

Game theory is a mathematical domain which aims at analyzing and predicting how rational players strategically behave in different situations. Basic principles assume all players first analyse what other players might do. This is denoted as strategical thinking. Depending on the result of the analytics and the panel of possible choices, players select the most suitable option. This phase is named optimization. Then players try to adapt their responses in order to reach a form of equilibrium where all players get the maximum profit they can have.

In this section, a formal definition of game theory will be presented, then an explanation about the behavioral aspect of game theory will be given as well as a brief introduction of decision making technique.

4.2.1 Overview of game theory

Game theory is a branch of mathematics which has been pioneered by J. Von Neumann and O. Morgenstern [189] for economic purposes. Later, the fields of applications widened to other fields like sociology and psychology, engineering, computer science and information technology. One of the main goals consists in determining optimum strategies for players using mathematical and logical tools with the aim of dealing with a given situation. Such situations can be represented as gain maximization for one or several players or as risk minimization.

Three main type of games can be found: pure conflict, also denoted as zero-sum games like chess; cooperation games, where all participants are supposed to choose and to implement their actions together. The third type is more complex because it is neither a cooperation game nor a conflict game: participants choose their actions separately. However, one player's behavior with other players is a mix of cooperation and competition. The essence of a game consists in the dependence of all player's strategies. Either a game is sequential, or simultaneous.

In sequential games, the main principle consists in looking ahead in order to evaluate the situation and then to reason back. Each player tries to understand how other players could respond to his current moves and thus to anticipate plausible reactions. Using

	Silence	Defect
Silence	$(3, 3)$	$(10, 0)$
Defect	$(0, 10)$	$(5, 5)$

Table 4.1: Prisoners' dilemna [151]. If A stays silent and B betrays, B goes free and A stays 10 years in prison. If B stays silent and A betrays, A goes free and B stay 10 years in prison. If both remain silent, both of them stay 3 years in prison. Otherwise, if both betrays each other, they stay in prison for 5 years. If A and B cooperate, they stay 3 years in prison. If not, either the betrayer is free or stays 5 years in prison. What is the best strategy? Dixit and Nalebuff offer alternatives like mixing moves, strategic moves, bargaining, concealing and revealing about this problem [43].

this information, he calculates the best choice that allows him to take an advantageous position. It has to put himself into his opponent's shoes because he cannot impose its own reasoning scheme to them. Sequential games are limited and have a finite number of moves. Each player has to determine how to achieve the best outcome in this limit of moves. For games like tic-tac-toe, it is easy to compute all strategies in advance. On the contrary, games like chess have a prohibitively large combinatorial number of possible sequences, making the resolution hard to perform. This is why a player (human being and computers) can only look for a few moves ahead for that game.

In simultaneous games, things are more tricky because such games involve a logical circle of reflexion. Players are acting at the same moment without having any clue about what other players are thinking. An approach able to deal with this type of games has been introduced by John Nash with the famous Nash equilibrium. Let us explain the idea behind this concept of equilibrium. A set of choices is determined for each player. All players play their own best strategy, which improves the outcome for themselves. It may happen that the best choice of one player leads to win the highest outcome. The played strategy is denoted as dominant strategy for this player. Of course, if one wins, another player loses. The strategy which leads to the lowest income is denoted as dominated strategy. The search for the equilibrium consists in eliminating any dominated strategies while all players have dominant strategies. However, an equilibrium might not lead to a collectivity positive outcome. In the case of the prisoner's dilemma, illustrated by Table 4.1: maximizing the best private interests leads to a negative outcome for each player. Because an individual's success in making profit depends on the choice of others, or, in the case of this dissertation, depends on where all tasks are currently mapped, leading this dissertation to focus on behavioral game theory.

4.2.2 Behavioral game theory

Two types of games, based on thinking, learning, feeling and sharing principle, can be pointed out in behavioral games: one-shot games and repeated games.

One-Shot Games

The thinking model is designed to predict behavior and to provide initial condition for any models of learning. It allows to favor one strategy above others. The strategy attraction settles the probabilities of choosing the most suitable strategy using a response function. Some commonly used notations will now be defined. For any player i, there are m_i sets of strategies. Each set of strategies is indexed by j. This means that strategy j of player i is denoted by s_i^j. The period or step of the game is represented by t. When $t = 0$, the corresponding attraction of a given strategy j and a given player i can be written as $A_i^j(0)$. When the focus is set on a player i, other players will be denoted as -i. During some period t, players i's payoff for choosing s_i^j is valued by equation:

$$A_i^j(t) = \pi_i(s_i^j, s_{-i}(t)) \ . \tag{4.1}$$

Camerer indicates that the probability of having a specific set of strategies of a player i at time t can be computed using equation 4.2. [28]. One can notice that this equation consists in computing the ratio of the attraction of the specific set of strategy on the sum of all m_i sets of strategies player i possesses. A λ factor is also introduced in order to regulate the response sensibility.

$$P_i^j(t+1) = \frac{e^{\lambda . A_i^j(t)}}{\sum_{k=1}^{m_i} e^{\lambda . A_i^k(t)}} \ . \tag{4.2}$$

One-shot games are massively used in games where the aim is to determine a winning sequence among others. Paying attention to behavior of other players has the only interest of establishing the winning sequence. If one-shot games are applied to the mapping problem, the goal would consist in determining a global configuration or sequence which minimizes the global mapping value. This means that all previous sequences have to be taken into account. This aspect invalidates the representation of the mapping as a one-shot game. This is the reason why repeated games will now be presented.

Repeated games

In repeated games, a game is played repeatedly, with the same players. Such games are also named stage games. Unlike in one-shot games, best shots may not lead to the best results. During the game, players mays cooperate at some turn or defect at others turns, leading to the use of a mixed strategy. The game can be either repeated during a finite number of steps or infinitely repeated. Some associated notations for this kind of games will now be defined. Let $i \in \{1, ..., n\}$ be the set of players. Let $a_i \in A_i$ be the possible actions of player i, and let π_i be the respective payoff. Due to the repetition aspect, this type of game also includes a notion of history h. Because ofa infinite number of different actions, the history is rarely the same. For each step or period t of the game, the possible actions available may vary. Equation 4.3 is a sequence of the past action profiles at the beginning of the period.

$$h_t = (a(1), a(2), ..., a(t-1)) \in H_t = A^{t-1} \ , \tag{4.3}$$

where A^{t-1} consists in the set of actions performed until $t-1$, t be the number time occurencies and H_t represents the set of possible histories and is represented by equation 4.4:

$$H = \cup_{t=1}^{\infty} H_t \ . \tag{4.4}$$

The payoff of player i is $\pi_i(a(t))$ for step t. Player i's strategy $s_i \in S_i$ can be defined as a mapping from H to A_i. A chosen strategy $s_i \in S_i$ for player i involves a different set of actions. The average payoff of a sequence of actions at a period τ can be computed using equation 4.5, where $\gamma \in [0, 1)$ is a discount factor.

$$\pi_i(s) = (1-\gamma) \sum_{\tau=1}^{\infty} \gamma^{\tau-1} \pi_i(a_\tau) \tag{4.5}$$

In this kind of games, it is very difficult to find a global Nash equilibrium. Yet, on a local sight, it is easier to determine a Nash equilibrium for each stage of the game. This aspect is denoted as Subgame Perfect Equilibrium.

In this thesis, the task affectation process can be viewed as a repeated game. The algorithm is the only player. This aspect is known as game against nature. Actions consist in choosing where tasks should be mapped. The strategy consists in how to choose the right task to map. The global history is represented by the final global mapping and a period is designated by the number of tasks which have already been mapped. Now that the game environment has been defined, the next step consists in determining how to select the best task mapping, meaning which decision process could lead to the best possible mapping. This requirement allows the introduction of decision theory.

4.2.3 Decision theory

Decision theory is a very diversified field. Many different ways exist to theorize decisions. The different levels and ways of decision processes can be modeled by the following:

- punctual decisions;
- decisions based on past evolutions of one situation;
- decisions based on future outcomes;
- decisions based on behavior of people;
- decisions based on different choices;

Decision theory has been introduced in the middle of the 20^{th} century. Since then, economists, statisticians, psychologists, politicians, social scientists and even philosophers have been developing this field using very different approaches and tools transforming this research field into a very diversified and multipurpose domain. A way to model decisions consists in building a decision tree as illustrated by Figure 4.1. It is defined as a classification procedure that recursively partitions data sets into smaller sets. The subdivision is made with a rule that is defined by decision makers. A decision tree

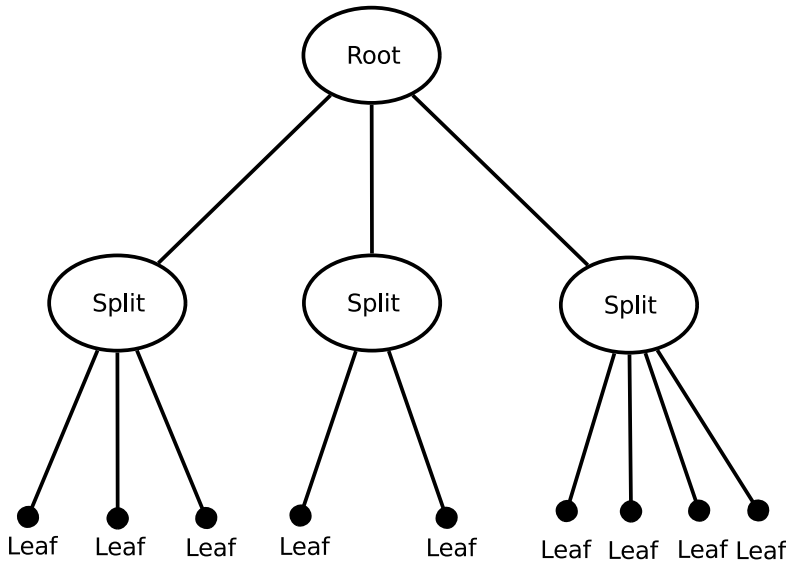


Figure 4.1: A decision tree. Each circle is a node. Data are split into smaller groups depending on a rule defined by the decision maker. At the end of the tree, each leaf contains a various amount of data.

is composed of a root node, several split nodes and terminal nodes denoted as leaves [159]. Despite of these disparate approaches, decision theory can be split into two main categories: Normative Decision Theory (NDT) and Descriptive Decision Theory (DDT). NDT consists in how decisions should be made. DDT consists in how decisions are made. In addition to all these aspects, the decision process is sequential.

Many research have been performed in order to model and partition the decision process into phases. Condorcet was the first to divide the decision process into three phases [40]. In the first phase, important principles which are mandatory in the well-understanding of the problem are identified. In the second phase, each player's suggestions are taken into account and considered. In the last phase, the choice between all alternatives is made. A more recent and precise decision process model has been developed by Brim *et al.* [24]. The decision process is divided in six steps:

1. Problem identification.
2. Necessary information search.
3. Determination of possible solutions.
4. Evaluation of each determined solution.
5. Selection of the suitable solution.
6. Implementation of the solution.

For a connection of decision process and this thesis, only step 4 which evaluates the solution will be focused on. It occurs that the largest part of all merged fields of

literature about decision making is dealing with this step. The definition that is given is only applicable to decision process in game theory and cannot be generalized to any other fields.

Solutions that have to be evaluated have been provided by utility functions of the modeled problem. One problem, which appears related to the mapping problem but does not have the same requirements and is not the same problem, is the minimax problem. In this algorithmic problem, costs are minimized and profit is maximized. A notion of uncertainty, defined by persistence of the choice and its impact on the final solution has to be dealt with. A good approach able to manage this incertitude is regret theory, because its aim is to eliminate or at least reduce the risk of choosing a solution that might be regretted afterwards.

4.3 Regret theory

Regret is a negative emotion that people feel when they realize that things would have been better if an action had been performed differently. The comparability of a decision outcome with the outcomes forgone is the central element in regret. Simonson [169] states that:

> " *regret and responsibility should be regarded as separate constructs. Regret represents the sorrow over something done or not done, regardless of whether the decision maker was responsible for the outcome[...]. The magnitude of responsibility, on the other hand, represents the degree of self-blame (or self congratulation) for the decision that led to the obtained outcome.* "

To fully understand the impact of this emotion and to be able to master it, it is important to penetrate into the psychology of this emotion and to discern the processes that may moderate it. Von Djik and Zeelenberg [186] studied some differences between factual and counter-factual outcomes. They determined that, depending on the context, the perception and the comparison motivation, the regret feeling is more or less strong.

Economist decision theoretician like Bell [16] and Loomes and Sugden [119] got inspired by this emotion. They provided a more mathematical definition of regret which could be used in decision processes. Sugden splits regret in two major components: it can firstly be interpreted as a wish that you had done differently; and secondly as a feeling of self blame [180]:

> "*The pain you feel when you compare "what is" with "what might have been" depends upon something more than the nature of two consequences you are comparing. It seems to depend also on the extent to which you can defend your original decision to yourself as reasonable, sensible or normal[...]the intensity of regret depends on the extent to which the individual blames himself for his original decision.*"

Loomes and Sugden introduced a regret theory under uncertainty model where players have to choose between pairs of choices determined by a mathematical representation of

the outcome of actions. The regret model applied to the mapping problem will be detailed later in this thesis. Several types of regret can be identified. Each type corresponds to one problem to solve.

4.3.1 External regret, internal regret and swap regret

When an algorithm uses policy π_1, it sometimes incurs some loss in terms of quality. Unfortunately, if this algorithm had used another policy π_2 to deal with the problem, the loss would have been less. Then, in this case, regret value corresponds to the difference between π_1 and π_2. This raises an interrogation: how can different policies be categorized and how to know which policy should be used. Blum and Mansour provided some elements of response. Regret can be split into three parts: external regret, internal regret and swap regret [19].

External regret is the highest level approach. It considers all alternative policies that are independent of the choice made by the algorithm. It is also denoted as *best expert problem*. This means, solution qualities of the algorithm are compared to the best of N actions in retrospect. Loss is compared to a constant sequence of possible solutions. Hannan is one of the first that emphasizes this aspect [76]. Hannan-consistency property guarantees, in a long run, that players have an average payoff as large as the best reply payoff resulting from the empirical distribution of actions of other players. External regret is mostly used in on-line algorithms. Obtained performance may match with that of an optimal static off-line algorithm modeling all possible static solutions. Some famous external approaches result from a work of Foster and Vohra where a randomized strategy for selecting better forecasts is applied without making any assumptions about events distribution or errors distribution [58].

Now, the focus is set on internal regret. In contrast to external regret, the loss is compared to a class of sequences. That is to say, a single simple modification performed in the algorithm. For every occurrence, a given action i is converted into another action j. The aim is to maximize payoffs. One of the first approaches is performed by Blackwell on the *Von Neumann* minimax theorem which is a zero-sum game between two players. Adding some features of the law of large numbers, assertions are made about the possible extension that each player can control the center of gravity of the actual payoffs in a long series of plays. A center of gravity is a quantitative method able to determine the distribution center location that will minimize costs. The question this work answered deals with a given $r \times s$ matrix $M = \|m(i,j)\|$ [187], whose $m(i,j)$ elements are points of an euclidean N-Space. It uses a notion of approachability in order to prove that the center of gravity of all payoffs is in or near a subset $S \subset N$-Space [18]. This approachability was taken in other approaches like Hart and Mas-Colell where a correlated equilibrium of the game is determined using adaptive strategies described in [79]. Cesa-bianchi and Lugosi, on their side, improve and find out performance bounds in decision theory field. They also establish a link between potential based analysis in learning algorithm and their counterpart developed in game theory. Internal regret appears in the latest [30]. Foster and Vohra also introduced notions of internal regret as an alternative to external decision processes. Instead of measuring the success of a decision scheme by comparing

it to other schemes, internal sequence will be transformed [57].

In some circumstances, low external regret can be transformed into low internal regret. This transformation is called swap regret and has been proposed by Blum and Mansour [19]. It allows to simultaneously swap multiple pairs of actions. This means, for N actions, swap-regret is bounded by N times internal regret. In a repeated game scenario, if each player uses an algorithm which determines an action, whose regret computation is sublinear to the number of time steps, the empirical distribution of actions of players converges to a correlated equilibrium. Moreover, this approach can be used either in either full information model or in partial information model.

Now that several form of regrets have been presented, the focus is laid on the first version of regret established by Loones and Sugden; based on Bell's work.

4.3.2 Formal definition

Regret theory was first proposed by Bell [16]. Later, Loones and Sugden presented many works about this theory of choice under incertitude, other way to define regret theory, which can describe several aspects.

Loones and Sugden regret approach

Regret theory is always formulated by a choice between two actions (A_1 and A_2) made by a player. Of course, the choice condition is under uncertainty. The player does not know which of a number of states of the world will be performed. Using the definitions of Leonard Savage [162], a world is an object about which a person is considered and a state of the world is defined by a full description of the world, without leaving any relevant detail undefined. In some states, A_1 will provide a better consequence or payoff, while in others, it will be A_2 which will lead to a better consequence. Now, let us assume that the player chooses action A_1, but unfortunately, the consequence which occurs is worse than the one that would have occurred had he chosen A_2. The player will feel regret of not having chosen the best choice. Otherwise, he would have feel joy. In this theory, players can foresee if an action will lead to regret or joy. This information will obviously affect the selection made by the player [181].

Let the world be a infinite set of states. Depending on the rules which are used, the probability that a state occurs instead of another may be uniform or depend of a unknown distribution which can or cannot be identified. Let a state be denoted as $S_j = \{S_1, ..., S_n\}$ where n can be infinite. Most of the time, operations are computing in a finite subset of S. When choosing an action, S is unknown. For each action, there will be a various number of consequences which will arise. A consequence is denoted as X. An action A is composed of a finite number k of consequences $A_i = \{x_{i1}, x_{i2}, , , , x_{ik}\}$. Because there is only the choice between two actions, $A_1 = \{x_{11}, x_{12}, , , , x_{1k}\}$ and $A_2 = \{x_{21}, x_{22}, , , , x_{2k}\}$. If player chooses A_1 and state S_j occurs, the associated consequence will be x_{1j}. He also knows that consequence x_{2j} would have occurred if he had chosen A_2.

Utility stands for the motivation of players in any game. Utility function represents, for a given player, a numerical value assignment for all possible outcomes of the game.

This assignment requires that a higher number implies a more preferred outcome. In regret theory, utility (or level of satisfaction) of having a consequence x_{1j} and missing out x_{2j} is represented by $v(x_{1j}, x_{2j})$. ρ is the function which provides a real-valued utility which evaluates the utility for each pair $(x_{1j}, x_{2j}) \subset X^2$. Loones and Sugden formulate the utility using the following functional form:

$$v(x_{1j}, x_{2j}) = C(x_{1j}) + R(C(x_{1j}) - C(x_{2j})) \ . \tag{4.6}$$

where $C(x_{1j})$ and $C(x_{2j})$ are the corresponding utilities of x_{1j} and x_{2j}. R corresponds to the function which measures the regret. In most games, the goal of each players is to maximize v. The level of satisfaction utility computation function can be expressed in many different ways. Quiggin uses a more simple approach [150]. The set of consequence X is ordered by preference relationship. The level of satisfaction is defined by the following equation:

$$v^*(x_{1n}, x_{2n}) = v(x_{1n}, x_{2n}) - (x_{2n}, x_{1n}) \ . \tag{4.7}$$

It displays the net advantage because v is not decreasing in its first argument and not increasing in the second argument. The positive expectation or mean of v^* is provided by:

$$E[v^*(x_{1n}, x_{2n})] = \sum_{n=1}^{N} \rho_n v^*(x_{1n}, x_{2n}) \ . \tag{4.8}$$

In equation 4.8, the notation ρ_n is introduced. It corresponds to a probability measure. Because we are in a model which works under uncertainty, this assumes the use of probabilities. Let s_i be one of a N possible state of the world occurring with probability $\pi(s_i)$. Depending on the game features, the distribution π may change. Probability also appears for the finite given n-tuple set of consequences X. In [121], probability distribution of consequences is denoted as prospect. A prospect (or risky prospect) is denoted as $p_i = (p_{i1}, p_{i2}...., p_{ik})$ where p_i corresponds to the probability of observing consequence x_i. Moreover, probabilities are stochastic, which means $\sum p_i = 1$. A choice set s_i often consists of any number of prospects. That is to say, the consequence probability may interfere in the state selection. Let $\rho(p_1, s_1)$ be the probability of choosing s_1 and $\rho(p_2, s_2)$ the probability of choosing s_2. If

$$\sum_{i=0}^{k} p_{1i} u(x_{1i}) \leq \sum_{i=0}^{k} p_{2i} u(x_{2i}) \ , \tag{4.9}$$

then $\pi(p_1, s_1) = 0$ and $\pi(p_2, s_2) = 1$ [122].

Loomes and Sugden indicate, using equations 4.7, 4.8 and 4.9, that statewise stochastic dominance is preserved for all pairwise choices [120]. In addition, Quiggin shows that stochastic dominance will hold for A_i over A_j if, and only if, for any state k such that $x_{ik} \leq x_{jk}$, it exists another equally probable state l such that $x_{jl} \leq x_{ik} \leq x_{jk} \leq x_{jl}$ [149].

4.3.3 Use of regret theory in the literature

In everyday life, one must often make decisions in benign situations which tend to repeat frequently. For instance, which queue is the fastest in the supermarket? What route to

drive in order to reach the destination as fast as possible? These decisions are made in an uncertain environment where many decisive factors are unknown. In order to increase difficulty, other players with unknown strategies are also part of the environment, which can be described as repeated games, and often play against us. Blum and Mansour understood very well the situation and proposed a model which can describe it. They also propose regret-based approaches on full or partial information games which may allow to reach a Nash Equilibrium [20].

Reaching the Nash equilibrium using regret approaches has been subject to many investigations for the no-regret learning field. An algorithm is considered as no-regret if, for every input sequence, the regret grows sub-linearly in T periods which correspond to the history of the repeated game [115]. Greenwald and Jafari defined a general class of no-regret algorithms able to span the spectrum from no-internal regret learning to no-external regret algorithm. A class of game-theoretic equilibrium, denoted as Φ, is also defined which shows that any distribution of execution of no-regret algorithm converges to this equilibrium [72],[88]. Germano and Lugosi introduced the regret testing which allows the player to use stochastic learning defined by Foster and Young [59] to reach Nash equilibrium without knowing anything about the opponent, even if it exists [67]. However, regret theory is not only used in order to provide interesting approaches, it is also used in order to determine bounds on learning procedures using Markov Decision Processes [7], or in more precise algorithmic problems: regret Matching algorithms [73] and Distance Constrained Vehicle Routing problem [61], among others.

Another aspect to consider, matrix games are often used in the literature, but much less is known about convex games [167]. A convex game is based on a convex set of functions. In this type of games, one way to win the game consists in being part to a powerful cooperation of players. A coalition with a high number of players is more attractive than a feeble numbered coalition. This means, when players are attracted by a coalition, the attraction factors proportionally increases. This aspect only works when the game is played cooperatively leading to a "snowballing" effect. For this type of game, traditional regret algorithm cannot be applied. It has to be adapted as was done in [81] and [70].

Regret algorithm is also often used in optimization problem which are under uncertainty like random sampling [123], spanning tree, shortest path [9], p-center problems [8], cheapest insertion for TSP [80] and many others. Most of the cited problems are dealing with the minimax regret problem introduced by Von Neumann [187].

In this dissertation, tasks are placed one after another. The previous choices have to be taken into account and a will to optimize the global solution quality is present. This scheduling problem can be related to the mapping problem in some particular cases. Because there is currently no regret algorithm on the mapping problem, the focus will be laid on regret algorithm for the scheduling problem. A number of minimax regret scheduling problems are dealt in [110]. For instance, Averbakh use the regret for maximum weighted tardiness scheduling problem [8]. Kaspersky and Lu *et al.* also worked with regret for scheduling single machine sequencing [101], [102] and [124].

One can notice that regret algorithm are able to propose acceptable solutions for scheduling problems. Because scheduling and mapping problems are similar, regret theory

will now be applied on the mapping problem in order to minimize the solution quality of the mapping.

4.4 Regret Based Heuristic for mapping of non-unitary weighted task graphs

In the previous chapter, the Task Wise Placement (TWP) algorithm (Algorithm 5) has been introduced. Although the introduced algorithm provides satisfying results on unitary weighted instances (*i.e.* all weights and resources are set to 1), it is not able to deal with instances with various weights. Therefore, after a quick redefinition of TWP, a new regret-based algorithm will be presented, able to provide valuable solutions for this issue.

4.4.1 Task-Wise Placement behavior on non-unitary task graphs

Brief summary of TWP

In TWP, for the evaluation of the cost of mapping a task onto a node, a metric has been set up called the *distance affinity*. It is also described in the previous chapter. In a single phase placement process, TWP iteratively assigns each task with respect to this metric.

Initially, a task whose number of adjacent edges is maximum is placed on an arbitrary node. All its (unassigned) neighbors are placed into a waiting set and their corresponding distance affinities towards the selected node are computed.

Next, the task with the highest task to node affinity is selected and removed from the waiting set. If two or more tasks have the same affinity, the priority corresponds to the FIFO order. The selected task is then placed in the node with whom it has the greatest affinity.

When the chosen node is saturated, all tasks whose greatest affinity corresponds to a saturated node are removed from the waiting set. All unsaturated nodes which are in the neighborhood of the saturated node are assigned one unassigned task in the order of a pre-generated ordering, generated by Breadth-First Traversal (BFT).

Bounds of TWP

The general behavior of the algorithm with homogeneous instances is that a task may and will always be assigned to the node with sufficient free space and with the best affinity. On non-unitary instances, diverse weight values constrain the execution of the algorithm to a further extent. In some configurations, the mapping cannot be computed because of insufficient free space on the target node. Previous choices, which lead to the actual task configuration in a node, may prohibit the mapping of a task which could have really improved the solution quality. This behavior tends to dramatically decrease the quality of the mapping solutions.

4.4.2 Introduction of the task cost notion

In TWP, in order to determine to which node a task should be mapped, a *distance affinity* function between a task t and a node n is used. This function is defined as follows:

$$\text{aff}(t,n) = \sum_{n'} \sum_{t'} x_{t'n'} q_{tt'} \times \frac{1}{2 \times d_{nn'} + 1} \ . \tag{4.10}$$

Each time a task is mapped, the affinities of all tasks which can be affected by this mapping have to be computed again. In the new algorithm, the notion of task to node affinity is replaced by a notion of task to node cost. The cost function is the following:

$$\text{cost}(t,n) = \sum_{n'} \sum_{t'} x_{t'n'} q_{tt'} d_{nn'} \ . \tag{4.11}$$

Maximizing the affinity corresponds to minimizing the cost. This new function corresponds to the local cost added to the global DPN mapping objective function detailed in Subsection 1.7 when t is assigned to n. It has the advantage of removing a compute expensive division operation, with positive impact on global execution performance.

4.4.3 A new task selection model based on regret theory

In the traditional regret approach, the choice is only made between the two best nodes. Limiting the mapping selection to only two nodes which present the lowest cost at the time of the current task selection potentially misses important opportunities which may lead to disastrous solution quality. Kilby's augmented regret approach limits this behavior.

Kilby's augmented regret

Several efficient regret-based heuristics have been developed in order to improve the two choices model [3]. Kilby proposes a new approach which bypasses those bounds [104]. He assumes that sometimes, the best choice may be the third or the fourth element. By evaluating only the two most interesting choices, the best choice may be missed. The probability for this case augments when the number of choices is large. Moreover, in our case, bad choices lead to bad mappings due to the high level of task correlation. In Kilby's heuristic, the regret value is computed using all candidate nodes.

For each task t, an ordered set $C = \{c_{t1}, c_{t2}, ..., c_{tm}\}, m \leq |N|$ is built, containing the cost values to each of the candidate nodes (*i.e.* nodes onto which the task can be placed) in ascending order.

The regret value is now the average of all levels of satisfaction for all pairs of nodes. It can be expressed as follows:

$$regret_t = \frac{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (c_{tj} - c_{ti})}{\frac{m(m-1)}{2}} \ . \tag{4.12}$$

This formula can be computed in linear time using the equivalent expression:

$$regret_t = \frac{\sum_{i=1}^{m} c_{ti} \times (2i - m - 1)}{\frac{m(m-1)}{2}} \quad . \tag{4.13}$$

The task with the highest level of satisfaction is mapped to the node with the lowest corresponding task to node cost.

4.4.4 Description of the algorithm

Algorithm 6 performs the regret computation and Algorithm 7 is the regret-based mapping algorithm.

The choice of the first task to be mapped is similar to TWP (that is, the task with the highest sum of adjacent edge bandwidths). The task is assigned to the first available node. Typical processor architectures show a sufficient level of symmetry for us to assume the initial task assignment has no impact on mapping quality. Then, the regret criterion explained in Section 4.4.3 is used for iteratively choosing the next task to be mapped, and this process is repeated until all tasks are mapped. A set of candidate tasks is maintained along this process, containing all the adjacent tasks that have not been mapped yet and are neighbors of the already assigned tasks; this set corresponds to the tasks that have a non-zero regret value.

4.4.5 Complexity of the algorithm

The complexity of the whole algorithm is now studied.

The regret computation of Algorithm 6 can be divided in three parts. First, the cost values of each node where the task fits in are pushed into a queue. It is obviously useless to deal with costs of nodes where the task can never be mapped onto. This selection phase is performed in $\mathcal{O}(|N|)$. Second, costs are sorted in ascending order with typical sort complexity $\mathcal{O}(|N| \log |N|)$. Third, the regret equation 4.13 is computed. This computation has a complexity of $\mathcal{O}(|N|)$. The greatest complexity of the function correspond to the sort, this means the computation of Algorithm 6 has a complexity of $\mathcal{O}(|N| \log |N|)$.

Algorithm 7 can be divided into 2 parts: initialization and main loop. In the initialization part, the task to node mapping costs for all neighbors of the assigned task are computed along with the corresponding regret values. The complexity of this operation is $\mathcal{O}(\delta(n + |N| \log |N|)) = \mathcal{O}(\delta |N| \log |N|)$.

Now, we focus on the main loop part of the algorithm. All tasks are processed sequentially and for each of them, the following operations are performed: a search for the task with the greatest regret value ($\mathcal{O}(|T|)$), a search for the node with the minimum task to node cost value ($\mathcal{O}(|N|)$), a cost update operation in the same way as in the initialization with an identical complexity of $\mathcal{O}(\delta |N| \log |N|)$. Hence, assuming $|T| > |N|$, the main loop complexity (which is the overall complexity) is $\mathcal{O}(|T|^2 + |E||N| \log |N|)$.

Algorithm 6 compute Regret

Input: t ▷ The task whose regret has to be computed.
Input: $G' = (N, E')$ ▷ Node graph.
Input: $cost_t$ ▷ node cost vector of task t of size $|N|$.
Input: A ▷ Task mapping assignment array of size $|T|$.
Input: $C \in \mathbb{R}$ ▷ Node capacity.
Output: $regret$ ▷ Regret value of t.

```
Q ← {∅}                                          ▷ Task to node costs array.
regret ← 0                                       ▷ Regret value.
for all n ∈ N do
    if w_t + Σ_{t':A[t']=n} w_{t'} ≤ C then
        Q ← cost_{tn}
sort(Q)                                          ▷ Ascending sort of Q
k ← −|Q| + 1
for all i ∈ Q do
    regret ← regret + Q[i] × k
    k ← k + 2
if |Q| > 1 then
    regret ← regret / (|Q|(|Q|−1)/2)
else
    regret ← 99999999
return regret
```

Algorithm 7 Regret-based Task Placement

Input: $G = (T, E)$: task graph with q bandwidth matrix
Input: $G' = (N, E')$: node graph with d distance matrix
Input: $C \in \mathbb{R}$: node capacity
Output: A: Task mapping assignment array of size $|T|$

$R \leftarrow \{0\}^{|T|}$ ▷ Regret value of each task
$W \leftarrow \{\varnothing\}$ ▷ Waiting set
$cost_t \leftarrow \{0\}^{|N|}$ ▷ task to node cost vector of task t
$t_0 \leftarrow \underset{t}{\text{argmax}}(\sum_{t' \in N_G(t)} q_{tt'})$
$A[t_0] \leftarrow 1$
for all $t \in N_G(t_0)$ **do** ▷ Update of the cost vector
  $W \leftarrow W \cup \{t\}$
  for all $n \in N$ **do**
    $cost_{tn} \leftarrow cost_{tn} + q_{t_0 t} \times d_{1n}$
  $R_t \leftarrow computeRegret(t, G', cost_t, A, C)$
while $|W| > 0$ **do**
  $t \leftarrow \underset{t \in W}{\text{argmax}}(R_t)$
  $n \leftarrow \underset{n \in N}{\text{argmin}}(cost_{tn})$
  $A[t] = n$
  $W \leftarrow W \backslash \{t\}$
  for all $t' \in N_G(t)$ **do** ▷ Update of the cost vector
    $W \leftarrow W \cup \{t'\}$
    for all $n' \in N$ **do**
      $cost_{t'n'} \leftarrow cost_{t'n'} + q_{tt'} \times d_{nn'}$
    $R_{t'} \leftarrow computeRegret(t', G', cost_{t'}, A, C)$
return A

Now that the complexity has been determined, the algorithm is compared to other approaches in terms of performance.

Algorithm 8 GRASP procedure

Input: $G = (T, E)$ ▷ Task graph with q bandwidth matrix.
Input: $G' = (N, E')$ ▷ Node graph with d distance matrix.
Input: $C \in \mathbb{R}$ ▷ Node capacity.
Input: Max ▷ The maximal number of times the GRASP is performed.
Output: A ▷ Task mapping assignment array of size $|T|$.

1: **for all** $(seed \in [0; Max])$ **do**
2: | $A_c \leftarrow construction\_phase(G, G', C, seed)$
3: | $A \leftarrow local\_Search\_Phase(A_c)$
4: **return** A

4.5 A greedy randomized adaptive search procedure for the Regret-Based Approach

On certain instances, a greedy algorithm may be trapped by a wrong decision which leads to a lower mapping quality independently of the size of the instance. In order to avoid this trap, one solution consists in running several times a randomized version of the algorithm and only considering the best generated-solution. One efficient randomization approach consists in performing a Greedy Adaptive Search Procedure (GRASP) [49].

4.5.1 Definition of the GRASP procedure

A GRASP is a two-phase iterative process: a construction phase and a local search phase. During the construction phase, a list of possible solution is built and the benefits of all elements in this list are computed. This computation makes the GRASP procedure adaptive. A solution is randomly chosen among all solutions in the list. Once the solution has been built, a local search phase occurs. The neighborhood of the solution, obtained during the construction phase, is explored and if a better solution if found, it replaces the current solution. This phase is performed until no better solution can be determined. Algorithm 8 sums up the description of the GRASP procedure applied to our approach.

Let us emphasize the fact that the execution of all the different randomized procedures are strictly independent, and can be done in parallel, meaning that with a sufficient number of processors, the GRASP procedure only takes the longest time among all the different searches.

Feo *et. al.* use this approach in order to solve the set covering problem and the maximal independent set problem which are combinatorial problems [49].

Sirdey *et. al.* propose a GRASP-based approach in order to generate partitions which, applied to a simulated annealing heuristic able to solve the QAP, solves the mapping

problem as explained in Section 2.4. In this heuristic, only the construction phase is performed, but no local search.

Stan *et. al.* elaborate a GRASP method able to determine the best partitioning-based mapping with routing constraint. [176], [177], [178].

4.5.2 GRASP and RBA

The GRASP [1] procedure is applied to the RBA heuristic. The construction phase consists in running Algorithm 7 as many times as necessary in order to build the best possible mapping. However, RBA is modified during the determination of the highest regret value operation detailed in Subsection 4.4.3 and illustrated by Algorithm 6. Several identical maximal regret values may exist. The heuristic, detailed above, used a FIFO policy for selecting the highest regret of the first task. Instead of choosing the first task, the choice is randomly made among the highest regret values. This choice can be extended to tasks whose regret values are not the highest but five or ten percents lower. Our implementation uses a uniform random generator. In order to avoid the generation of same random values, the seed of the random generator is different for each construction phase.

In this dissertation, no local search heuristics have been developed despite the fact that many heuristics which perform local search operation can be found in the literature, more precisely in multilevel partitioning or in mapping solvers cited in Chapter 2. However, most of them are processing a neighborhood search with load balancing constraint on nodes. This is the reason why these heuristics cannot be applied to our problem. The design of such a heuristic is at least as hard as the elaboration of the mapping heuristic. This is the reason why the GRASP is only performed using a construction phase.

4.6 Application of the heuristic

4.6.1 Execution platform

The target system is a server based on the AMD Opteron 6172 processor running at 2.1 GHz.

4.6.2 Instances

We challenged our algorithm to test its strength on several graph architectures. In the previous chapter, we only used two kinds of instances: grids and logic gate networks. In this work, we added more types of instances. Table 4.2 shows the list of all instances which have been computed for our tests. The provided diameter values are estimates obtained by computing the eccentricity of pseudo-peripheral tasks.

[1] Stricto-sensu our algorithm is a semi-greedy algorithm rather than a proper GRASP as it lacks the local search phase. Still, as the semi-greedy procedure is the key component of a GRASP, such algorithms are often slightly abusively referred to as GRASP in the operational research literature.

Graph	Name	$\|T\|$	$\|E\|$	diam. (est.)
Grids	grid100x100	10,000	39,588	200
	250,000	250,000	997,474	1000
LGN	b14	10,012	38,111	14
	b19	231,266	881,690	41
Mat.M	Cote/vibrobox	12,328	330,311	9
	Cote/troll	213,453	11,979,477	77
Series-Parallel	10235 SP40	10,235	31,470	34
	15779 SP30	15,779	44,244	347
	201880 SP40	201,880	620,594	196
	261033 SP30	261,033	732,097	1,155
Random	10000	10,000	68,808	8
	200001	200,001	11,883,611	4

Table 4.2: Task graphs with several topologies. Shown values are the number of tasks, the number of edges and the diameter.

Grids

It consists of a set of grid (or square mesh) shaped task topologies, which correspond to regular dataflow computational networks like matrix products.

Logic gate networks

The Logic Gate Networks (LGN) instances are generated out of logic gate networks resulting in the design of microprocessors. These configurations of task networks typically can be found in real life complex dataflow applications.

Matrix market

The Matrix Market [38] is a large collection of sparse matrices. The purpose of this collection is to be used in comparative studies of algorithm. Each matrix has got its dedicated web page where its features can be found. Besides those features, a graphical representation of the matrix and the corresponding graph can be found. It is also possible to download it. One major quality of the Matrix Market collection is the presence of very large sparse matrices.

We specialized our study on one famous collection of matrices, the Walshaw's instances [195]. Those matrices can also be found in the Matrix Market Collection. Those matrices are often used in the literature for other works like graph partitioning [11], [86].

Random graphs

We made the assumption that random generated graphs do not show the particular local structure which makes the dataflow task graphs convenient for mapping, meaning that

if our algorithm is able to find a good mapping on random topologies, it may be able to find a good mapping on any kind of graphs.

They are generated using the Erdos-Renyi algorithm properties [46]. This choice has been made because it is one of the most famous models for generating random graphs. The generation is such that we preserve an average degree similar to that of typical dataflow graphs.

Series-parallel graphs

Series-parallel graphs present a structure very similar to that of classical DPN graphs. They are fertile testing ground for various hypotheses like modeling sequences of events in time series data [126], modeling transmission sequencing requirements of multimedia presentation [4] or the modeling of task dependencies in a dataflow model of massive data processing for computer vision [34].

Two classes of instances have been randomly generated with two different generation settings. One of these classes, named SP30, contains two instances of $15,779$ and $261,033$ tasks respectively which have been generated using a probability of 30% of parallel branches. The other class, named SP40, with two instances of $10,235$ and $201,880$ tasks with a parallel branch probability of 40%.

4.6.3 The node layout

The node layout is a square torus, hence the number of nodes in all instances is a square value.

For each pair of nodes (n, n'), the distance $d_{nn'}$ is the Manhattan distance between nodes n and n'. It can be modulated to improve the efficiency of the algorithm like for instance using the square Manhattan distance values than the standard one.

4.6.4 Random weight generation

We want to adapt the tasks and the edges weights in order to work with instances which are very similar to real applications.

Let x a random variable for weight generation. In order to be as close as possible to real applications, the weight distribution should follow a continuous probability distribution for each realization of x. The most suitable distribution to this requirement is the Normal (Gaussian) distribution, whose density function is detailed below:

$$f(x, m, s) = \frac{1}{s \times \sqrt{2\pi}} e^{(\frac{-1}{2})(\frac{x-m}{s})^2} \quad . \tag{4.14}$$

In Equation 4.14, m represents the average value and s corresponds to the standard deviation.

Our test cases consists of several types of tasks which show different characteristics (for instance in terms of memory usage or CPU time consumption). In this study, we generated instances using two different task types with significantly different behaviors.

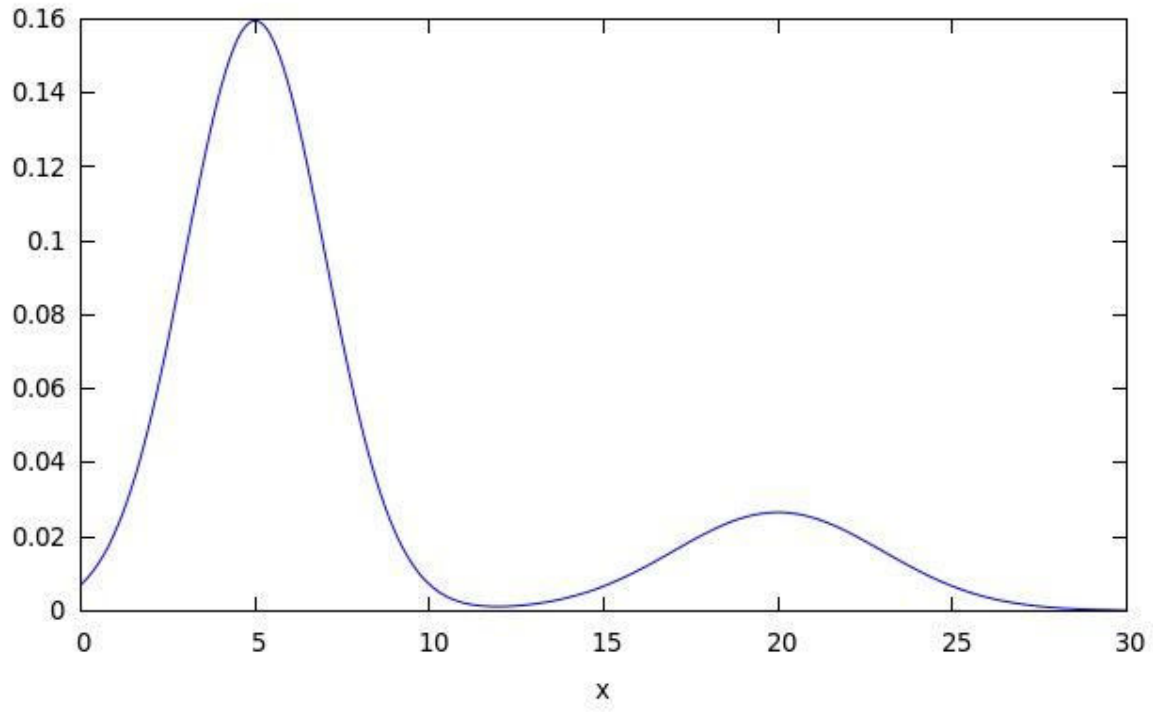


Figure 4.2: weight density function ($m_0 = 5, s_0 = 2, m_1 = 15, s_1 = 3$).

Each of those behaviors follow the same normal distribution but with different parameters; for this probabilistic model, we use the bimodal density function below:

$$g(x) = \alpha f(x, m_0, s_0) + \beta f(x, m_1, s_1) \ . \tag{4.15}$$

In Equation 4.15, m_0 and m_1 respectively have been assigned the values 5 and 15. s_0 and s_1 are the corresponding standard deviations whose values are respectively 2 and 3. We do not want the x value to be too irregular, this is why the range of x values is: $x \in [0 : 30]$. α and β respectively have the value of 0.8 and 0.2 because in usual dataflow task graphs, more small tasks are found than large ones.

Figure 4.2 provides a graphical view of the weighting density function.

4.6.5 Experimental protocol

We suppose that, depending on the number of nodes, the behavior of the algorithm is different. This is why the algorithm is applied to several node values: 16, 36, 64, 144, 256, 400, 576, 1024. Those values have been chosen because they correspond to square torus architectures. The upper bound limit of the number of nodes is set to 1024, which is a large value compared to the size of currently existing manycore systems.

In order to reduce statistical anomaly, for each number of nodes and each task graph topology, a group of 30 instances has been generated. Each instance in a same group shares the same topology and the sum of all task weights are the same. The individual

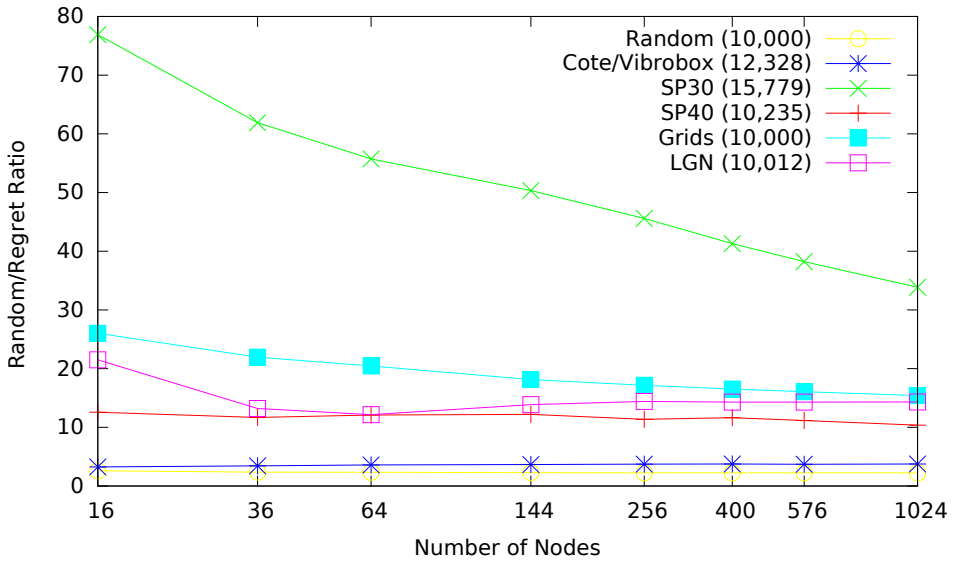


Figure 4.3: $\frac{Random}{Regret}$ for task graphs with an order of magnitude of $10,000$ tasks.

# of nodes	10,000 Grids		12,328 MM		15,779 SP30	
	ratio	time(s)	ratio	time(s)	ratio	time(s)
16	26.02	0.9	3.25	14.43	76.90	0.77
36	21.95	1.08	3.44	15.77	61.87	1.00
64	20.45	1.29	3.58	17.83	55.72	1.35
144	18.13	1.81	3.66	23.16	50.34	1.82
256	17.15	2.64	3.72	31.1	45.60	2.65
400	16.52	3.79	3.76	42.34	41.30	3.67
576	16.06	5.24	3.71	55.56	38.22	4.97
1024	15.43	8.87	3.75	89.6	33.87	8.38

Table 4.3: Ratio $\frac{Random}{Regret}$ for task graphs with an order of magnitude of $10,000$ tasks.

weights are different from one instance to another. Those weights are generated using the density function from Equation 4.15. In order to add some stress on the algorithm, the total sum of task weights is equal to 95% of the sum of node capacities, making the instances very tight with a very reduced feasible domain. The following results in this paper are average values, for each group, of the results obtained of the 30 corresponding instances.

4.6.6 Experimental results and analysis

In this analysis, the algorithm which generates a random mapping is denoted as Random Approach (RA). The algorithm with a regret-based task selection is called Regret Based Approach (RBA). The solution value of the random approach is divided with that of the

# of nodes	10,235 SP40		10,000 random		10,012 LGN	
	ratio	time (s)	ratio	time (s)	ratio	time (s)
16	12.56	4.39	2.58	11.36	21.51	6.5
36	11.69	4.76	2.36	12.2	13.21	6.89
64	12.09	4.34	2.35	12.48	12.17	7.17
144	12.20	4.95	2.27	13	13.86	8.19
256	11.36	5.75	2.26	16.45	14.42	9.11
400	11.64	6.45	2.26	18.6	14.29	10.9
576	11.17	7.48	2.26	22.33	14.31	13.12
1024	10.37	10.03	2.25	30	14.13	17.71

Table 4.4: Ratio $\frac{Random}{Regret}$ for task graphs with an order of magnitude of 10,000 tasks.

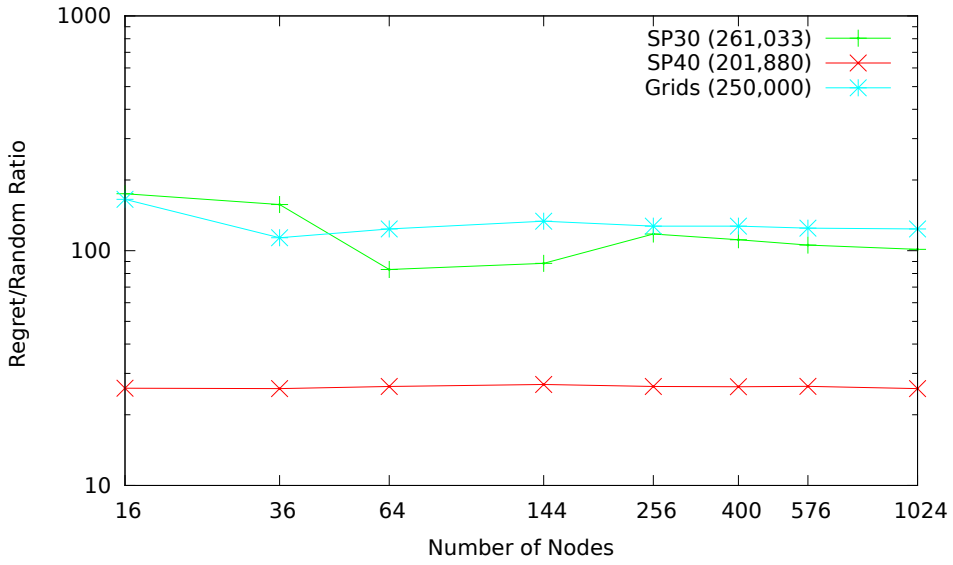


Figure 4.4: $\frac{Random}{Regret}$ for task graphs with an order of magnitude of 200,000 tasks.

# of nodes	231,266 LGN		200,001 rand		201,880 SP40	
	ratio	time (s)	ratio	time (s)	ratio	time (s)
16	7	3,886	1.57	14,889	21	1,474
36	8	3,483	1.43	13,562	25	1,189
64	9	3,000	1.43	14,000	27	1,119
144	9	3,041	1.38	13,526	27	1,165
256	10	2,686	1.36	12,281	27	1,214
400	10	2,255	1.51	12,790	27	957
576	10	2,655	1.37	13,165	27	942
1024	9	2,567	1.37	15,156	26	1,107

Table 4.5: Ratio $\frac{Random}{Regret}$ for task graphs with an order of magnitude of 200,000 tasks.

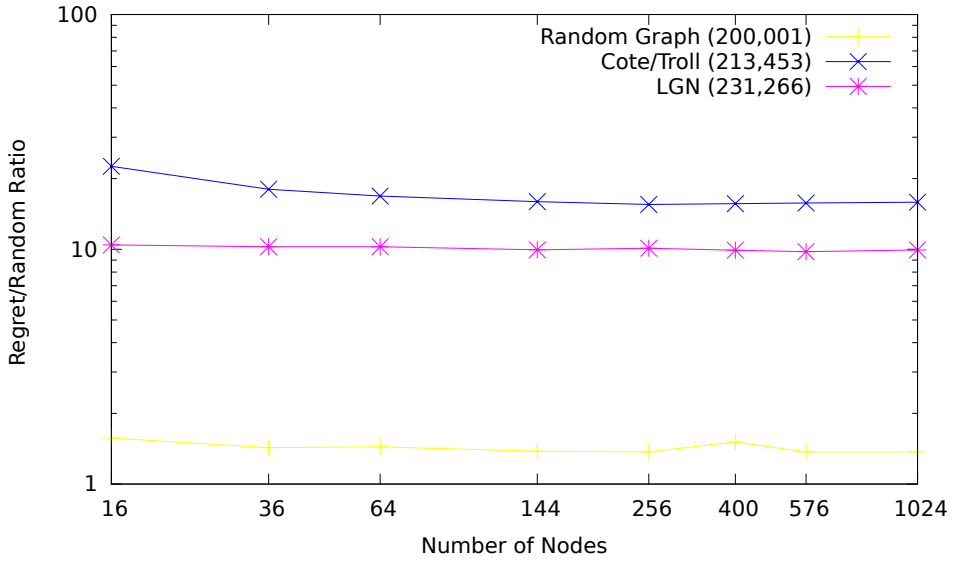


Figure 4.5: $\frac{Random}{Regret}$ for task graphs with an order of magnitude of $200,000$ tasks.

# of nodes	250,000 Grids		261,033 SP30		213,453 MM	
	ratio	time (s)	ratio	time (s)	ratio	time (s)
16	158	1091	144	62	23	829
36	92	698	143	59	18	825
64	104	592	84	47	17	939
144	114	756	86	65	16	1,055
256	117	872	115	74	16	1,260
400	119	825	118	109	16	1,570
576	119	815	107	75	16	1,934
1024	123	821	102	113	16	2,875

Table 4.6: Ratio $\frac{Random}{Regret}$ for task graphs with an order of magnitude of $200,000$ tasks.

# of nodes	1,000,000 Grids		1,256,931 SP30		2,285,274 SP30	
	ratio	time (s)	ratio	time (s)	ratio	time (s)
16	847	3,743.00	4,667	1,743.00	411	17,090.00
36	402	5,381.00	3,044	1,887.00	458	13,728.00
64	186	5,666.00	2,124	1,792.00	295	16,482.00
144	196	7,133.00	2,419	1,944.00	322	14,882.00
256	262	7,635.00	3,224	2,191.00	430	14,965.00
400	243	8,118.00	2,874	2,281.00	439	14,098.00
576	237	8,506.00	2,837	2,505.00	429	13,448.00
1024	232	8,927.00	2,625	2,848.00	398	14,702.00

Table 4.7: Ratio $\frac{Random}{Regret}$ for task graphs with an order of magnitude of 1 and 2 million tasks.

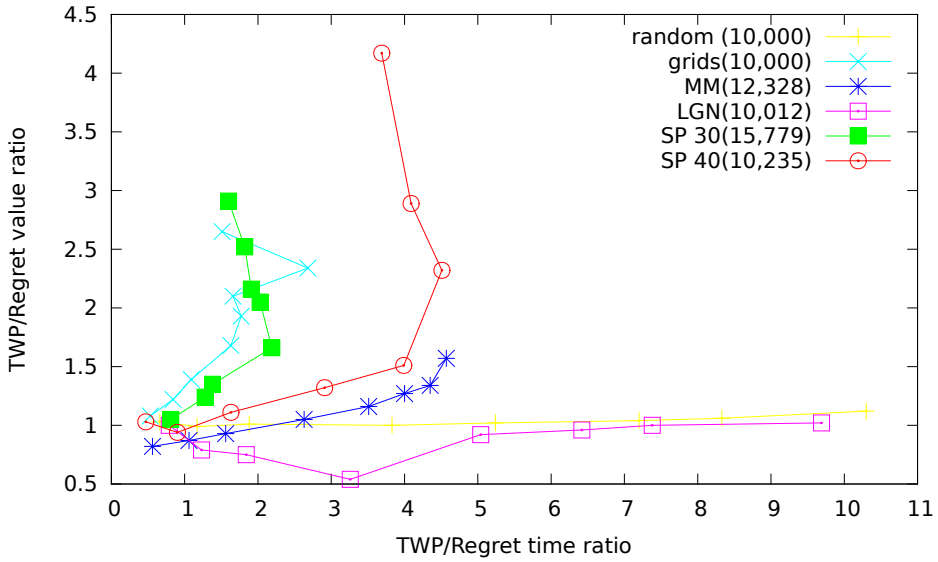


Figure 4.6: Ratio $\frac{TWP}{Regret}$ for task graphs with an order of magnitude of $10,000$ tasks.

regret-based approach. The aim of this ratio is to evaluate the quality of the mapping according to the metric defined in section 1.8. As explained in section 1.7, as our problem is a minimization problem, higher solution quality ratios are preferable.

Table 4.3 and Table 4.4 display the solution quality ratio of the RBA compared to RA on instances with tens of thousands of tasks. Figure 4.3 is a graphical view of these tables. The values of the abscissa axis correspond to the number of nodes and the values of the ordinate axes correspond to the ratio. Table 4.5, Table 4.6 and Figure 4.5 show a similar analysis on instances of hundreds of thousands tasks.

Random graphs ($10,000$, $200,001$) present no particular structures to exploit. This makes it difficult for the regret-based approach to exploit any internal graph structures. However the average results of RBA are 1.5 to 2.3 times better than RA's for any number of nodes. Moreover, from 16 to 256 nodes, it is the approach which is the longest to compute.

The sparse matrix of the matrix market graph (12,328) is a sparse matrix in which all non-zero values are situated in very heterogeneous positions. It still contains enough local structures that can be exploited to show better behavior than on random graphs (average ratio of 3). Nevertheless, starting from 256 nodes, computation times increase very significantly and is twice as high as those of random graphs.

For Logic Gate Networks ($231,266$), the average ratio is around 9. The average mapping time is around 1 hour.

Grids (10,000, 250,000) present a particular square architecture which is easily exploitable. On this kind of architecture, our algorithm starts to provide respectively about 16 and 150 times better results than RA. This is very encouraging because this type of structure has similarities with some DPN architectures.

For series-parallel class SP40, the obtained results provide average ratio values which

vary from 11 to 27. Besides, on SP30 instances, results vary from 33.26 to 144. This is the only task graph topology for which we can notice that the ratio decreases when the number of nodes increases.

Another aspect to mention concerns computation times. For grids and series-parallel architectures, the computation is very fast. We are able to find a good valid mapping in less than 5 seconds for tens of thousands tasks and less than 2 minutes for hundreds of thousands tasks.

Our regret-based method shows particularly good behavior when the instances contain a large part of local structure that can be exploited, similarly to real life dataflow applications. This is particularly true on grids, logic gate networks and series-parallel task graph topologies. The best results are obtained on graphs with high diameter value and relatively low average degree such as grids and series-parallel. The impact of the diameter is visible when comparing SP30 and SP40 instances for which a higher diameter leads to better solution values and better performance. On LGN graphs, where the diameter is significantly lower than our series-parallel instances, the obtained ratios are still up to ten times better than RA.

Figure 4.6 is a graphical comparative study between the task-wise placement method and the regret-based approach on task graphs with an order of magnitude of $10,000$ tasks. The abscissa values correspond to the $\frac{TWP}{Regret}$ time ratio and the ordinate axis value correspond to the $\frac{TWP}{Regret}$ mapping solution value ratio. Each curve on the figure corresponds to the mapping of one of the task graph topology groups on the different target architectures (from 16 to 1024 nodes). For each curve, the ratios tend to increase when the number of nodes increases. Three different behaviors can be observed. For grids and SP30, the solution quality of RBA tends to be much better than TWP on the larger target architectures while the computation time of RBA remains slightly better (ratio < 2). For LGN and random instances, the solution quality ratio is close or inferior to 1 while the computation time ratio increases along with target size (RBA is 10 times on 1024). SP40 and Matrix Market show an intermediate behavior where both ratios increase with the target size.

These results show that the regret-based approach performs in a much more scalable way than TWP and provides better solution qualities on graphs with the most similarity with typical dataflow process network graphs. Due to excessive run times of TWP, this study could not be performed on graphs with orders of magnitude of $200,000$ tasks.

In a completely different subject, the RBA heuristic is applied on task graphs up to 1 and 2 millions of tasks. Table 4.7 displays the obtained results. In line of what we concluded, RBA is able to map grids in less than 3 hours with solution values which are several hundreds of times better than the random approach. However, over 1 million, grids computational time start to explode. The focus is now set on series-parallel task graphs. RBA is able to map within one hour a one million task graphs with solution values several thousands times better than those from the random approach. Moreover, on task graphs up to 2 million tasks, it takes 4 to 5 hours for mapping the task graph with a ratio still hundreds of times better than the random approach.

# of solutions	0%	1%	2%	5%	10%	20%
1	17.2	32.4	41.9	60.4	82.9	97.4
2	23.6	43.3	52.6	71.8	90	99.2
3	29.3	50.2	59.9	78.7	93.6	99.5
4	34	55	64.5	83.3	94.9	99.7
5	38.3	59.3	68.4	86.6	96.4	99.8
6	42.3	62.8	71.9	88.4	96.9	99.9
7	46.3	66.3	74.8	90	97.3	99.9
8	49.6	69.1	77.1	91.5	97.9	99.9
9	52.3	71.2	78.8	92.7	98.3	99.9
10	55.3	73.4	80.5	93.4	98.4	99.9

Table 4.8: Cumulated percentage of solution values.

4.6.7 Experimental results and analysis using GRASP procedure

The experimental protocol of applying the GRASP procedure on the RBA, as explained in Subsection 4.5, consists in the following: the seed of the random generator is modified for each mapping generation. We chose to select 30 different seed values because, in practice, it corresponds to a sufficiently large value which allows to assess the behavior of the heuristic. Depending on obtained mapping results, the best mapping solution quality and the worst execution time among 30 mapping solutions values are selected. However, one drawback of computing all seeds consists in the explosion of execution time. For instance, on Logic Gate Networks instances up to $231,266$ tasks, the execution times varies from $4,000$ seconds to $12,000$ seconds. In addition, empirical results show that instances which are long to compute tend to exhibit a poor solution quality. We cannot afford to compute all seeds in order to find the lowest mapping quality even if it is run in parallel. The question we want to answer is how many iteration of the GRASP procedure are required in order to get a mapping quality value as close as possible to the lowest mapping quality value. The terminology best solution quality means the lowest mapping value among all seeds. This terminology is used in the following.

Determination of the number of solutions

In order to address this issue, instances detailed in Subsection 4.6.2 are computed again. Random graphs up to $200,000$ tasks are not considered because execution times of this type of instances are too important. The same experimental protocol is performed than that described in Subsection 4.6.5. All types of tasks graphs are mapped onto several manycore architectures whose number of node vary. This means that for each type of graphs, 8 sets of instances are created. For each set, a group of 30 instances are generated with different tasks and edges weights. The average mapping value is computed. One additional feature is added. For each instance, the mapping heuristic is performed with all 30 different seeds in parallel.

# of nodes	10,000 Random		12,328 MM		15,779 SP30	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	1.00	0.78	1.01	0.83	1.25	0.64
36	1.01	0.85	1.00	0.85	1.13	0.69
64	1.02	0.84	1.01	0.89	1.08	0.81
144	1.01	0.81	1.00	0.94	1.06	0.92
256	1.00	0.92	1.00	0.98	1.05	0.96
400	1.00	0.92	1.00	1.03	1.03	0.98
576	1.00	0.97	1.00	1.06	1.06	1.04
1024	1.00	0.98	1.00	1.07	1.04	1.05

Table 4.9: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for task graphs with an order of magnitude of 10,000 tasks.

After computing all instances, the best mapping value (BMV) and the worst execution time are determined for each group of instances. During the computation of these thousands of instances, the order (in terms of execution time) at which the BMV occurs is determined. First column of Table 4.8 shows the result of this analysis. This table represents the percentage of the cumulated number of best quality solutions found at the corresponding number of generated solutions. In this table, only the first 10 generated solutions among the 30 seeds is displayed. Clearly, even if the event of having the BMV at the first generated solution is a probability of 0.17, after 10 generated solutions, there is only a probability of 0.55 to get the BMV. In order to reduce the number of generated solutions for getting an acceptable solution, mapping values which are 1 percent worse than BMV, are accepted. It happens that after 3 generated solutions, the probability of having one acceptable solution is 0.5. We choose to increase the range of acceptable solutions by considering solutions which are 2%, 5%, 10% and 20% worse than BMV. For 2% worse solutions values, it requires only 2 generated solutions for having a probability of 0.5 to get an acceptable solution. For 5% worse solution values, 7 generated solutions are needed to get a probability of 0.9 for having one acceptable solution. For 10%, it takes 2 generated solutions for having the same probability and for 20% worse solutions, just one generated solution is needed.

GRASP computation

Table 4.9 and Table 4.10 display the results of the GRASP by accepting first only best solution values. In these tables can be found solution quality ratios and execution time ratios of $\frac{RBA}{GRASP}$ on instances of thousands of tasks. Table 4.11, Table 4.12 show a similar analysis on instance of hundreds thousands of tasks. If solution quality ratios are lower than 1, it means that solution qualities of the GRASP procedure are worse than that obtained by RBA computations. In addition, if time ratios are lower than 1, it means that GRASP computations take more time than RBA computations.

On random graphs of several thousands tasks, the solution quality ratios is 1 for all sets of instances. Time ratios vary from 0.78 to 0.98. It has been explained that these

# of nodes	10,235 SP40		10,000 grids		10,012 LGN	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	1.09	0.86	1.16	0.74	0.99	0.73
36	1.13	0.87	1.13	0.78	0.98	0.76
64	1.06	0.77	1.10	0.81	1.02	0.76
144	1.01	0.85	1.12	0.94	1.04	0.79
256	1.08	0.88	1.12	0.98	1.05	0.87
400	1.01	0.89	1.12	1.04	1.04	0.94
576	1.05	0.93	1.10	1.08	1.04	0.99
1,024	1.07	0.95	1.08	1.09	1.03	0.99

Table 4.10: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for task graphs with an order of magnitude of 10,000 tasks.

# of nodes	231,266 LGN		201,880 SP40	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	1.09	0.54	1.00	0.46
36	1.06	0.57	1.06	0.49
64	1.09	0.51	1.06	0.39
144	1.10	0.5	1.06	0.38
256	1.09	0.57	1.08	0.63
400	1.10	0.6	1.07	0.60
576	1.09	0.58	1.07	0.57
1,024	1.09	0.29	1.08	0.65

Table 4.11: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for task graphs with an order of magnitude of 200,000 tasks.

# of nodes	250,000 Grids		261,033 SP30		213,453 MM	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	0.9	0.6	1.08	0.64	1.09	0.71
36	1.03	0.71	1.04	0.67	1.07	0.71
64	1.06	0.7	1.06	0.67	1.08	0.75
144	1.04	0.73	1.09	0.73	1.09	0.79
256	1.10	0.81	1.06	0.75	1.09	0.89
400	1.10	0.81	1.07	0.81	1.09	0.94
576	1.10	0.82	1.06	0.79	1.06	0.97
1,024	1.11	0.85	1.06	0.87	1.06	1.00

Table 4.12: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for task graphs with an order of magnitude of 200,000 tasks.

types of graphs are very hard to map. Using a GRASP does not improve the solution quality. Moreover, GRASP computation takes more time. These computational times are already important, computing GRASP on several hundreds of thousands tasks is pointless.

On Logic Gate Networks, solution quality ratios increase from 2% to 5%. However, for several thousands of tasks, time ratio varies from 0.73 to 0.99. Time variations seems more important for a weak number of nodes than on an important one. This trend concerns only this order of magnitude. For several hundred thousands of tasks, the solution quality increase remain constant around 10% but the ratio is around 0.6 for all nodes except for 1024 which runs slower.

On Matrix Market instances, as detailed in Subsection 4.6.5, these matrices present similar behavior than random graphs. There is no improvement on solution quality and, for instances lower than 256 nodes, it runs slower. However, starting from 400 nodes, GRASP starts to run faster. It might be that for large target topology, it is much easier for GRASP to map this type of graphs.

On grids, solutions are 1.03 to 1.17 better than RBA. One exception can be found on several hundreds of thousands tasks mapped on 16 nodes where solution quality is lower. This may be due to the fact that GRASP was not able to find a better solution than RBA. on several thousands tasks graphs, starting from 256 nodes, GRASP runs 4 to 9% faster than RBA. However, for hundreds of thousands tasks, time ratio varies from 0.6 to 0.85.

On SP40 instances, independently of the number of tasks, GRASP provides better solutions. Moreover, for $10,235$, solution quality ratios vary from 1.01 to 1.13 while the corresponding time ratio varies from 0.77 to 0.93. For $201,880$, solution quality ratios are slightly better (from 1.00 to 1.08) but computational time ratios are lower than $10,235$ (0.38 to 0.63).

On SP30 instances, for $15,779$, solution qualities vary from 1.03 to 1.25 However, for a 25% increase of the solution quality, execution time of this solution is 49% lower than the deterministic heuristic. From 36 nodes to 400 nodes, execution time are lower however, starting from 576 nodes, GRASP runs faster than RBA. On $261,033$ instance, solution qualities are also slightly better (from 4 to 8%) while execution times are lower (from 0.64 to 0.87).

After analyzing all instances, we observe best solution quality performance is reached for series-parallel graphs (SP30 and SP40) ad grids meaning that GRASP is efficient and consistently improves the solution quality. Random graphs, Matrix Market and LGN remain hard to map. This is the reason why solution quality increase is weak. In addition to this analysis, independently of the type of instance, we can also notice that while the number of nodes increase, GRASP execution time decrease. Depending on the type of instance, standard deviation between 16 and s$1,024$ nodes ratios are not the same.

Partial GRASP computation

We decided to accept solutions which are 5% worse than BMV and stopped GRASP execution after 7 generated solutions. With this number of generated solutions, the

# of nodes	10,000 Random		10,000 grids		12,328 MM	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	0,97	0,81	1,08	0,88	0.97	0,84
36	0,97	0,85	1,070	0,92	0.97	0,85
64	0,97	0,85	1,041	0,95	0.97	0,89
144	0,97	0,88	1,05	1,04	0.97	0.94
256	0,97	0,91	1,06	1,05	0.97	0.96
400	0,97	0,92	1,04	1,09	0.97	1.01
576	0,97	0,96	1,04	1,10	0.97	1.03
1,024	0,97	0,98	1,00	1,09	0.97	1.04

Table 4.13: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for partial GRASPs for task graphs with an order of magnitude of 10,000 tasks.

# of nodes	10,012 LGN		15,779 SP30		10,235 SP40	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	0,96	0,79	1,15	0,72	1,04	0,86
36	0,94	0,80	1,06	0,81	1,075	0,88
64	0,97	0,80	1,02	0,93	1,01	0,88
144	0,99	0,85	1,00	1,03	0,97	0,86
256	0,99	0,89	1,00	1,04	1,03	0,89
400	0,99	0,94	0,98	1,03	0,97	0,89
576	0,99	0,98	1,00	1,07	1,00	0,93
1,024	0,99	0,98	0,93	0,99	1,01	0,93

Table 4.14: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for partial GRASPs for task graphs with an order of magnitude of 10,000 tasks.

# of nodes	213,453 MM		231,266 LGN		250,000 Grids	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	1,02	0,87	1,03	0,66	0,87	0,72
36	1,01	0,84	1,01	0,70	0,95	0,78
64	1,02	0,93	1,02	0,70	1.00	0,76
144	1,05	0,95	1,04	0,70	0,97	0,81
256	1,04	1,04	1,03	0,75	1,03	0,87
400	1,04	1,07	1,05	0,79	1,03	0,88
576	1,02	1,07	1,04	0,73	1,03	0,88
1,024	1.01	1.06	1,03	0,48	1,04	0,90

Table 4.15: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for partial GRASPs for task graphs with an order of magnitude of 200,000 tasks.

# of nodes	201,880 SP40		261,033 SP30	
	Sol. Qual.	Time Ratio	Sol. Qual.	Time Ratio
16	0,94	0,61	0,99	0,70
36	1,01	0,67	0,95	0,77
64	0,99	0,61	0,95	0,80
144	1,00	0,56	1,01	0,88
256	1,01	0,81	1,00	0,90
400	1,03	0,78	0,99	0,95
576	1,02	0,72	0,99	0,93
1,024	1,03	0,84	0,97	0,96

Table 4.16: Solution Quality and Time Ratios $\frac{Regret}{GRASP}$ for partial GRASPs for task graphs with an order of magnitude of 200,000 tasks.

probability of 0.9 to get an acceptable solution. Table 4.13 and Table 4.14 display solution quality and execution ratios of GRASP procedure compared to RBA on instances of thousands of tasks. Table 4.15, Table 4.16 also show a similar analysis on instance of hundreds of thousands tasks. Even if it is a GRASP procedure which is performed, this GRASP execution is denoted as Partial GRASP.

Because obtained solutions correspond to a subset of solutions, the behavior of the GRASP remains the same than the one described in Subsection 4.6.7. The focus will be set on differences between the whole set of seeds and a subset of seeds.

First of all, the best solution correspond to a mapping solution which is 5 percent worse than MBV. The range of possible solutions is brighter and the solution quality can be less than 5% worse. Table 4.8 shows that for 7 generated solutions, the probability of having BMV is 0.46, for 1% worse solution this probability is 0.66 and for 2% worse solution , the probability is 0.75. Solution quality ratios displayed by Table 4.13, Table 4.14, Table 4.15 and Table 4.16 are in this range. Despite all these probabilities, it occurs that for one instance, 7 generated solutions did not lead to a good solution quality. This corresponds to the bold value in Table 4.13.

Even if solution qualities are near from BMV, all of them are lower. However, if execution time ratios are analyzed, one can notice that either partial GRASP runs faster than GRASP or it has the same computational time.

Using a GRASP procedure allows to determine acceptable solution qualities with feeble deterioration which can be selected as shown in Figure 4.8. The number of generated solutions and the computational time depends on the chosen deterioration. By allowing an increasing percentage of worse solutions, it occurs that the GRASP procedure is able to find an acceptable solution faster and needs to generate a feeble number of solutions. This procedure can run in parallel and once a solution which fits the deterioration criteria is found, the GRASP computation can be stopped.

4.7 Conclusion

The goal of this chapter was to propose a new heuristic method able to tackle large non-unitary weighted instances of the DPN mapping problem with capacity constraints which emerge from the cyclo-static dataflow parallel programming paradigm. Being able to provide good placements is crucial for execution performance of large-sized dataflow programs on massively parallel architectures which are currently emerging. A previous approach we called TWP provided average quality results on homogeneous instances but not on non-unitary weighted instances. In this chapter, a greedy regret-based approach adapted from TWP is presented. The cost property computation improvement led to better runtime and a new regret-based task selection approach allows an adaptive mapping taking further advantage from the locality properties of the task graph architecture.

As there is no known possible way to compare our results with exact solutions and as we could not find equivalent algorithms in the literature to compare our results with, we used the metric defined in Section 1.8 in order to evaluate the performance of our heuristic. Moreover, TWP has been adapted for non-unitary weighted tasks and used as a comparison heuristic. These two comparisons allowed us to appreciate the performance of RBA.

Finally, we choose to apply a GRASP procedure on RBA. A uniform random task selection has been added in the task selection process. After performing experiments, we were able to build a statistical table which indicates the number of solution generation needed in order to get an acceptable solution as close as possible to the best solution that RBA is able to provide. Depending on the required solution quality, we are able to estimate how many parallel executions are needed to be performed in order to achieve this solution quality.

Due to the fact that the series-parallel architecture is very similar to the DPN architecture and that our algorithm provides solutions of good quality on such topologies, we may conclude that our method is able to map large non-unitary weighted task graphs on SMPs under capacity constraints in scalable time.

Conclusion

The purpose of this thesis is the development of static mapping methods of tasks graphs on homogeneous architectures. Several requirements for the mapping have to be met. Firstly, the mapping algorithm should be scalable; secondly it has to be topology-aware of the target architecture, and finally, it has to enforce node capacity constraints. This mapping fits in the ΣC compilation toolchain and is intended to reduce inter-task communication while merging as many tasks as possible on the same node. The interest of this gathering consists in minimizing the execution time of the application.

In the state of the art, many mapping approaches are dealing with capacity constraints and are topology-aware of the target architecture. However, despite the efficiency of the produced mappings, these approaches are not scalable and their limits are often reached around several thousands of tasks. In addition, partitioning and mapping solver, able to deal with task graphs up to tens of million tasks, are focused on load balancing constraints and are not considering capacity constraints. This aspect might shows that the state of the art, at the time of writing of this thesis, does not provide approaches able to solve our problem. This leads us to the establishment of heuristics able to fulfill the aforementioned requirements.

The analysis of the state of the art shows, at the time of writing this thesis, that no comparable algorithm can be found. This led to the establishment of a metric which is able to evaluate the quality of the newly developed heuristics. This metric is based on the solution value obtained by a random mapping. This metric allows to locate results quality where the optimal is not known and where no approximations can be performed.

A first approach consists is solving the mapping problem using unitary-weighted tasks and edges. Two heuristics have been developed: Subgraph-Wise Placement (SWP) and Task-Wise placement (TWP). SWP is a two-phase mapping method. First, a subgraph is built using the breadth-first traversal algorithm. Second, the subgraph is mapped using a notion of affinity. The second heuristic is a greedy mapping heuristic which maps tasks one after another using a notion of distance affinity. This type of mapping is defined as one-phase mapping. These heuristics have been tested on grids and graphs which come from Logic Gate Networks (LGN). They have been compared to the random based mapping (RBM) metric and a heuristic which is currently used in the ΣC compilation toolchain,called Partitioning and Placing (P&P). For grids, on a small number of tasks (below thousands), SWP and TWP run faster but provide a lower solution quality than P&P. However, when the number of tasks increases, the solution quality of both heuristics becomes better than P&P. On LGN, only TWP provides better solution quality than P&P

and runs tens times faster. However, despite the fact that solution quality of SWP is lower than that of P&P, it runs several order of magnitude faster than P&P. For grids, the RBM metric shows that SWP and TWP tends to outperform P&P while the number of tasks increases. However, on LGN instance, SWP is not able to provide satisfying results. However, TWP is able to provide very good solution quality. We may conclude that SWP is more grid-oriented while TWP is more adaptable to different types of instances. The application domain of these heuristics only concerns unitary-weighted tasks graphs. This is why a more general heuristic is needed for non-unitary weighted tasks graphs.

A second work consists in solving the mapping problem using various weights for tasks and edges. Uncertainty, raises by the fact that an adequate mapping during the execution of the algorithm may lead to poor solution quality, is managed by regret theory. This sub-domain of game theory is able to achieve good results for choices under uncertainty. However, it is not often used for mapping problems. It is more used in scheduling or quadratic assignment problems. The idea of applying it to the mapping problem comes from the fact that scheduling and QAP problems are close to the mapping problem. In the mapping heuristic, denoted as Regret Based Approach (RBA) which is also a one-phase mapping which places one task after another, tasks noticed during the computation are placed in a waiting set. Regret-theory is used at this step of the heuristic, more precisely in the determination of the task to map. It also provides the most suitable node onto which the selected task is to be mapped. The heuristic is applied on several task graph topologies such as grids, LGN, series-parallel (similar to dataflow process networks), sparse matrices and random graphs. This heuristic is compared to a non-unitary weighted adaptation of TWP and the RBM metric is used in order to evaluate the heuristic. Solution qualities of RBA is tens to hundreds times better than those of RBM for grids and series-parallel. Moreover, runtime of the heuristic on this type of topologies are the lowest compared to other topologies. On the contrary, random graphs or LGN which are very different to DPN, ration of solution qualities are less honorable but RBA is still better than RBM. By comparing RBA to TWP, it appears, depending on the task graph topology, that either RBA is faster than TWP but the solution quality is similar, or the solution quality of RBA is better than the solution quality of TWP but there is no significant speed-up. Finally, a parallel approach for the RBA has been set up. A greedy randomized adaptive search procedure (GRASP) is applied to the RBA. We managed to determine how many parallel runs of the RBA are needed in order to get the best parameterized solution value.

During this thesis, three heuristics which are scalable, topology aware of the target architecture and respectful of capacity constraints, have been developed. Moreover, a random-based metric has been introduced in order to compare the solution quality of these heuristics. However, despite of a solution quality which is acceptable, even good, the optimal mapping is not reached, meaning the solution quality is improvable. This leads to further investigations.

The focus has been set on the construction of the best mapping solution. These solutions can be improved by local-search heuristics. Many elements in the literature use local-search heuristics in their refinement techniques in order to improve the solutions. Such an approach can be found in many solvers like Metis and Scotch. However, many refinement heuristics focus on load balancing constraints. This aspect makes difficult

to find refinement techniques that could be applied to our problems. Many different strategies such as traditional local-search heuristics or more complex heuristics can be applied in order to improve our solutions. One approach consists in gathering nodes in pairs and for each pair to perform task swaps in order to improve the global solution. However, in that case, the refinement approach may be divided into two problems: finding an heuristic able to perform feasible task swaps while enforcing all capacity constraints and determining the sequence of pair of nodes which leads to the improvement of the solution.

One investigation consists in surpassing the 2 million task graphs limit and to be able to map these graphs onto target architecture including more than 1024 nodes. One way to overcome this limit and to reduce execution time consists in the parallelization of costs, affinities and regrets computations. For that, the use of hybrid CPU/GPU, in order to speed-up costs and regret computations, may reduce the global execution time of our heuristics.

Another aspect to consider consists in improving solution qualities for more dense application. During this thesis, we developed heuristics able to find good mapping qualities for sparse applications with series-parallel topologies. However, when the task graph is dense or has a more complicated topology, quality is decreasing. One approach would be to use a multi-start approach on tasks which are diametrically opposite and to map these tasks onto also diametrically opposite nodes of the target architecture. This approach will not provide equivalent performances but it may increases solution quality for dense applications.

Let us consider another perspective which is not related on the local improvement of our heuristics. Rather than mapping tasks graphs onto homogeneous architectures, the focus is now set on heterogeneous architecture. For architectures with different SMPs features, it is expected to have equivalent results because our heuristics takes into consideration available resources.

One last perspective consists in deepening the analysis of the random-based metric introduced in this thesis both from a theoretical and experimental viewpoint. In particular, we would like to experimentally verify if this metric still is of interest on other variants of the mapping problem and if it can be generalized to other problems and further investigate its theoretical properties and connections.

Finally we hope that our contribution was able to construct good mapping solutions which can be used as a basis for the design of local-search heuristics. Moreover, another contribution consists in the random based metric able to generate comparable mapping solutions for graphs with a large number of vertices. We also hope that this metric may be used as tool for comparable approaches in the mapping problem domain.

Bibliography

[1] Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[2] Tarun Agarwal, Ashok Sharma, and Laxmikant V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[3] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009.

[4] Paul D. Amer, Christophe Chassot, Thomas J Connolly, Michel Diaz, and Phillip Conrad. Partial-order transport service for multimedia and other applications. *IEEE/ACM Transactions on Networking (TON)*, 2(5):440–456, 1994.

[5] Françoise André and Jean-Louis Pazat. Le placement de tâches sur des architectures parallèles. 1987.

[6] Pascal Aubry, Pierre-Edouard Beaucamps, Frédéric Blanc, Bruno Bodin, Sergiu Carpov, Loïc Cudennec, Vincent David, Philippe Doré, Paul Dubrulle, Benoît Dupont De Dinechin, François Galea, Thierry Goubier, Michel Harrand, Samuel Jones, Jean-Denis Lesage, Stéphane Louise, Nicolas Morey Chaisemartin, Thanh Hai Nguyen, Xavier Raynaud, and Renaud Sirdey. Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor. In *Alchemy 2013 - Architecture, Languages, Compilation and Hardware support for Emerging ManyCore systems*, volume 18, pages 1624–1633, Barcelona, Spain, June 2013.

[7] Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in neural information processing systems*, pages 89–96, 2009.

[8] Igor Averbakh. Minmax regret solutions for minimax optimization problems with uncertainty. *Operations Research Letters*, 27(2):57–65, 2000.

[9] Igor Averbakh and Vasilij Lebedev. Interval data minmax regret network optimization problems. *Discrete Applied Mathematics*, 138(3):289–301, 2004.

[10] John Backus. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[11] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Soc., 2013.

[12] Stephen T. Barnard and Horst D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and experience*, 6(2):101–117, 1994.

[13] Kenneth E. Batcher. Design of a massively parallel processor. *Computers, IEEE Transactions on*, 100(9):836–840, 1980.

[14] Roberto Battiti, A Bertossi, and Romeo Rizzi. Randomized greedy algorithms for the hypergraph partitioning problem. *Randomization Methods in Algorithm Design, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 43:21–35, 1998.

[15] Roberto Battiti, Alan Bertossi, and Anna Cappelletti. Multilevel reactive tabu search for graph partitioning. *Preprint UTM*, 554, 1999.

[16] David E. Bell. Risk premiums for decision regret. *Management Science*, 29(10):1156–1166, 1983.

[17] Greet Bilsen, Marc Engels, Rud Lauwereins, and Jean Peperstraete. Cycle-static dataflow. *Signal Processing, IEEE Transactions on*, 44(2):397–408, 1996.

[18] David Blackwell et al. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[19] Avrim Blum and Yishay Monsour. From external to internal regret. 2007.

[20] Avrim Blum and Yishay Monsour. Learning, regret minimization, and equilibria. 2007.

[21] Shahid H. Bokhari. On the mapping problem. *Computers, IEEE Transactions on*, 100(3):207–214, 1981.

[22] Alessio Bonfietti, Luca Benini, Michele Lombardi, and Michela Milano. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pages 897–902. IEEE, 2010.

[23] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[24] Orville Gilbert Brim. *Personality and decision processes: Studies in the social psychology of thinking*, volume 2. Stanford University Press, 1962.

[25] Thang Nguyen Bui, Soma Chaudhuri, Frank Thomson Leighton, and Michael Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[26] Thang Nguyen Bui and Byung Ro Moon. Genetic algorithm and graph partitioning. *Computers, IEEE Transactions on*, 45(7):841–855, 1996.

[27] Rainer E Burkard. *Quadratic assignment problems*. Springer, 2013.

[28] Colin Camerer, Teck Ho, and Kuan Chong. Behavioral game theory: Thinking, learning and teaching. 2001.

[29] Ümit Çatalyürek and Cevdet Aykanat. Patoh (partitioning tool for hypergraphs). In *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer, 2011.

[30] Nicolo Cesa-Bianchi and Gábor Lugosi. Potential-based algorithms in on-line prediction and game theory. *Machine Learning*, 51(3):239–261, 2003.

[31] Jong-Sheng Cherng and Mei-Jung Lo. A hypergraph based clustering algorithm for spatial data sets. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 83–90. IEEE, 2001.

[32] Cédric Chevalier. *Conception et mise en oeuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problème numériques de très grande taille*. PhD thesis, Université Bordeaux I, 2007.

[33] Cédric Chevalier and François Pellegrini. Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework. In *Euro-Par 2006 Parallel Processing*, pages 243–252. Springer, 2006.

[34] Alok N. Choudhary, Bhagirath Narahari, David M. Nicol, and Rahul Simha. Optimal processor assignment for a class of pipelined computations. *Parallel and Distributed Systems, IEEE Transactions on*, 5(4):439–445, 1994.

[35] B Jack Copeland. *Colossus: The secrets of Bletchley Park's code-breaking computers*. Oxford University Press, 2006.

[36] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel computer architecture: a hardware/software approach*. Gulf Professional Publishing, 1999.

[37] V. David, C. Fraboul, J.-Y. Rousselot, and P. Siron. Étude et réalisation d'une architecture modulaire et reconfigurable: projet modulor. *Rapport technique*, 1:3364, 1991.

[38] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[39] Koen De Bosschere, Albert Cohen, Jonas Maebe, and Harm Munk. Hipeac vision 2015. 2015.

[40] Jean Antoine Nicolas Caritat de Condorcet. *Plan de constitution présenté à la Convention Nationale: les 15 et 16 février 1793, l'an II de la République: imprimé par ordre de la Convention Nationale*. Imprimerie Nationale, 1793.

[41] Marc Demange, Pascal Grisoni, and Vangelis Th Paschos. Differential approximation algorithms for some combinatorial optimization problems. *Theoretical Computer Science*, 209(1):107–122, 1998.

[42] K.D. Devine, E.G. Boman, L.A. Riesen, U.V. Catalyurek, and C. Chevalier. Getting started with zoltan: A short tutorial. In *Proc. of 2009 Dagstuhl Seminar on Combinatorial Scientific Computing*, 2009. Also available as Sandia National Labs Tech Report SAND2009-0578C.

[43] Avinash K Dixit and Barry J Nalebuff. *Thinking strategically: The competitive edge in business, politics, and everyday life*. WW Norton & Company, 1993.

[44] Paul Dubrulle and Renaud Sirdey. Génération d'un runtime de cadencement par un temps logique vectoriel. rapport technique dtsi. Technical report, SARC/09-138, Commissariata l'Énergie Atomique, 2009.

[45] Cagkan Erbas, Selin Cerav-Erbas, and Andy D Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *Evolutionary Computation, IEEE Transactions on*, 10(3):358–374, 2006.

[46] P ERDdS and A R&WI. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[47] Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

[48] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[49] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[50] Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349, 1987.

[51] Carlos Eduardo Ferreira, Alexander Martin, C .Carvalho de Souza, Robert Weismantel, and Laurence A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical Programming*, 81(2):229–256, 1998.

[52] Charles M. Fiduccia and Robert M. Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175–181. IEEE, 1982.

[53] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[54] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

[55] Matteo Fischetti, Michele Monaci, and Domenico Salvagnin. Three ideas for the quadratic assignment problem. *Operations research*, 60(4):954–964, 2012.

[56] Michael Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9):948–960, 1972.

[57] Dean P. Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1):7–35, 1999.

[58] Dean P. Foster and Rakesh V. Vohra. A randomization rule for selecting forecasts. *Operations Research*, 41(4):704–709, 1993.

[59] Dean P. Foster, H. Peyton Young, et al. Regret testing: Learning to play nash equilibrium without knowing you have an opponent. 2006.

[60] Pellegrini François. Contributions au partitionnement de graphes parallèle multiniveaux (contributions to parallel multilevel graph partitioning). 2009.

[61] Zachary Friggstad and Chaitanya Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 744–753. ACM, 2014.

[62] Keinosuke Fukunaga and Patrenahalli M Narendra. A branch and bound algorithm for computing k-nearest neighbors. *Computers, IEEE Transactions on*, 100(7):750–753, 1975.

[63] Lai-Wo Fung et al. Massively parallel processor computer, April 12 1983. US Patent 4,380,046.

[64] F. Galea and R. Sirdey. Méthode de cadencement d'applications flot de données cyclostatiques. *Rapport technique DACLE/10-070, Commissariata l'Énergie Atomique*, 2010.

[65] François Galea and Renaud Sirdey. A parallel simulated annealing approach for the mapping of large process networks. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1787–1792. IEEE, 2012.

[66] Michael R. Garey and David S. Johnson. Computers and intractability: a guide to np-completeness, 1979.

[67] Fabrizio Germano and Gabor Lugosi. Global nash convergence of foster and young's regret testing. *Games and Economic Behavior*, 60(1):135–154, 2007.

[68] Norman E. Gibbs, William G. Poole, Jr, and Paul K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):236–250, 1976.

[69] Kahn Gilles. The semantics of a simple language for parallel programming. In *In Information Processing'74: Proceedings of the IFIP Congress*, volume 74, pages 471–475, 1974.

[70] Geoffrey J Gordon, Amy Greenwald, and Casey Marks. No-regret learning in convex games. In *Proceedings of the 25th international conference on Machine learning*, pages 360–367. ACM, 2008.

[71] Thierry Goubier, Renaud Sirdey, Stéphane Louise, and Vincent David. σc: A programming model and language for embedded manycores. In *Algorithms and Architectures for parallel processing*, pages 385–394. Springer, 2011.

[72] Amy Greenwald and Amir Jafari. A general class of no-regret learning algorithms and game-theoretic equilibria. In *Learning Theory and Kernel Machines*, pages 2–12. Springer, 2003.

[73] Amy Greenwald, Zheng Li, and Casey Marks. Bounds for regret-matching algorithms. In *ISAIM*, 2006.

[74] Peter Hahn, Thomas Grant, and Nat Hall. A branch-and-bound algorithm for the quadratic assignment problem based on the hungarian method. *European Journal of Operational Research*, 108(3):629–640, 1998.

[75] Peter M Hahn, Bum-Jin Kim, Monique Guignard, J MacGregor Smith, and Yi-Rong Zhu. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications*, 40(3):351–372, 2008.

[76] James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3(97-139):2, 1957.

[77] James V. Hansen and William C. Giauque. Task allocation in distributed processing systems. *Operations research letters*, 5(3):137–143, 1986.

[78] J Pirie Hart and Andrew W Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

[79] Sergiu Hart and Andreu Mas-Colell. *A reinforcement procedure leading to correlated equilibrium*. Springer, 2001.

[80] Refael Hassin and Ariel Keinan. Greedy heuristics with regret, with application to the cheapest insertion algorithm for the tsp. *Operations Research Letters*, 36(2):243–246, 2008.

[81] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[82] Bruce Hendrickson and Robert Leland. The chaco user's guide: Version 2.0. Technical report, Technical Report SAND95-2344, Sandia National Laboratories, 1995.

[83] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

[84] Bruce Hendrickson and Robert Leland. A multi-level algorithm for partitioning graphs. 1995.

[85] John L. Hennessy and David A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[86] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. Engineering a scalable high quality graph partitioner. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[87] Jingcao Hu and Radu Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pages 233–239. ACM, 2003.

[88] Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *ICML*, volume 1, pages 226–233, 2001.

[89] Anil K Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[90] Nicholas Jardine and Robin Sibson. The construction of hierarchic and non-hierarchic classifications. *The Computer Journal*, 11(2):177–184, 1968.

[91] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[92] George Karypis. Multi-constraint mesh partitioning for contact/impact computations. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 56. ACM, 2003.

[93] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[94] George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 29. ACM, 1995.

[95] George Karypis and Vipin Kumar. Metis unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[96] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[97] George Karypis and Vipin Kumar. hmetis 1.5: a hypergraph partitioning package. Technical report, Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL http://www. cs. umn. edu/metis, 1998.

[98] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–13. IEEE Computer Society, 1998.

[99] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[100] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. *Version 1.0, Dept. of Computer Science, University of Minnesota*, 1997.

[101] Adam Kasperski. Minimizing maximal regret in the single machine sequencing problem with maximum lateness criterion. *Operations Research Letters*, 33(4):431–436, 2005.

[102] Adam Kasperski and Paweł Zieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97(5):177–180, 2006.

[103] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.

[104] Philip Kilby. The augmented regret heuristic for staff scheduling. *Proceedings of the 16th Australian Society of Operations Research. Notes*, 2001.

[105] David G. Kirkpatrick and Pavol Hell. On the completeness of a generalized matching problem. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245. ACM, 1978.

[106] Scott Kirkpatrick, C. Daniel Gelatt, Mario P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[107] Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: measurements, models, and methods. In *Computing and combinatorics*, pages 1–17. Springer, 1999.

[108] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[109] Tjalling C. Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, pages 53–76, 1957.

[110] Panos Kouvelis and Gang Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

[111] Marcio Kreutz, Cesar Marcon, Luigi Carro, Ney Calazans, Altamiro Susin, et al. Energy and latency evaluation of noc topologies. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 5866–5869. IEEE, 2005.

[112] Dominique LaSalle and George Karypis. Multi-threaded graph partitioning. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 225–236. IEEE, 2013.

[113] Edward Lee, Thomas M Parks, et al. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[114] Edward Ashford Lee and David G Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *Computers, IEEE Transactions on*, 100(1):24–35, 1987.

[115] Ehud Lehrer. A wide range no-regret theorem. *Games and Economic Behavior*, 42(1):101–115, 2003.

[116] Aris Leivadeas, Chrysa Papagianni, and Symeon Papavassiliou. Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1077–1086, 2013.

[117] Virginia Mary Lo. Heuristic algorithms for task assignment in distributed systems. *Computers, IEEE Transactions on*, 37(11):1384–1397, 1988.

[118] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[119] Graham Loomes and Robert Sugden. Regret theory: An alternative theory of rational choice under uncertainty. *The economic journal*, pages 805–824, 1982.

[120] Graham Loomes and Robert Sugden. Disappointment and dynamic consistency in choice under uncertainty. *The Review of Economic Studies*, pages 271–282, 1986.

[121] Graham Loomes and Robert Sugden. Some implications of a more general form of regret theory. *Journal of Economic Theory*, 41(2):270–287, 1987.

[122] Graham Loomes and Robert Sugden. Incorporating a stochastic element into decision theories. *European Economic Review*, 39(3):641–648, 1995.

[123] Antonio Lova and Pilar Tormos. Combining random sampling and backward-forward heuristics for resource-constrained multi-project scheduling. In *Proceedings of the Eight International Workshop on Project Management and Scheduling*, pages 244–248. Citeseer, 2002.

[124] Chung-Cheng Lu, Shih-Wei Lin, and Kuo-Ching Ying. Minimizing worst-case regret of makespan on a single machine with uncertain processing and setup times. *Applied Soft Computing*, 23:144–151, 2014.

[125] Chris Mack et al. Fifty years of moore's law. *Semiconductor Manufacturing, IEEE Transactions on*, 24(2):202–207, 2011.

[126] Heikki Mannila and Christopher Meek. Global partial orders from sequential data. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 161–168. ACM, 2000.

[127] Sorin Manolache, Petru Eles, and Zebo Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(2):19, 2008.

[128] César Marcon, André Borin, Altamiro Susin, Luigi Carro, and Flávio Wagner. Time and energy efficient mapping of embedded applications onto nocs. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 33–38. ACM, 2005.

[129] Cesar Marcon, Ney Calazans, Fernando Moraes, Altamiro Susin, Igor Reis, and Fabiano Hessel. Exploring noc mapping strategies: an energy and timing aware technique. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, pages 502–507. IEEE Computer Society, 2005.

[130] César Augusto Missio Marcon, Edson Ifarraguirre Moreno, Ney L.V. Calazans, and Fernando G. Moraes. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *Computers & Digital Techniques, IET*, 2(6):471–482, 2008.

[131] Radu Marculescu, Umit Y Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(1):3–21, 2009.

[132] Grant Martin. Overview of the mpsoc design challenge. In *Proceedings of the 43rd annual Design Automation Conference*, pages 274–279. ACM, 2006.

[133] Peter Marwedel, Jürgen Teich, Georgia Kouveli, Iuliana Bacivarov, Lothar Thiele, Soonhoi Ha, Chanhee Lee, Qiang Xu, and Lin Huang. Mapping of applications to mpsocs. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 109–118. ACM, 2011.

[134] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[135] Henning Meyerhenke, Burkhard Monien, and Thomas Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–13. IEEE, 2008.

[136] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Newsletter*, 20(3):33 – 35, 2006.

[137] Traian Muntean and El-Ghazali Talbi. Méthodes de placement statique des processus sur architectures paralleles. *TSI. Technique et science informatiques*, 10(5):356–373, 1991.

[138] Sameer Nene, Shree K Nayar, et al. A simple algorithm for nearest neighbor search in high dimensions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(9):989–1003, 1997.

[139] Heikki Orsila, Tero Kangas, Erno Salminen, Timo D. Hämäläinen, and Marko Hännikäinen. Automated memory-aware application distribution for multi-processor system-on-chips. *Journal of Systems Architecture*, 53(11):795–815, 2007.

[140] Ibrahim H. Osman and James P. Kelly. *Meta-heuristics: theory and applications*. Springer Science & Business Media, 2012.

[141] F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. SHPCC'94, Knoxville*, pages 486–493. IEEE, may 1994.

[142] F Pellegrini. Pt-scotch 5.1 user's guide. *Research rep., LaBRI*, 2008.

[143] François Pellegrini. *Application de méthodes de partition à la résolution de problèmes de graphes issus du parallélisme*. PhD thesis, 1995.

[144] François Pellegrini. A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. In *Euro-Par 2007 Parallel Processing*, pages 195–204. Springer, 2007.

[145] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[146] François Pellegrini and Jean Roman. Sparse matrix ordering with scotch. In *High-Performance Computing and Networking*, pages 370–378. Springer, 1997.

[147] Francois Pellegrini, Jean Roman, and Patrick Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Practice and Experience*, 12(2-3):69–84, 2000.

[148] Robert Preis and Ralf Diekmann. *The PARTY Partitioning-library: User Guide; Version 1.1.* Citeseer, 1996.

[149] John Quiggin. Stochastic dominance in regret theory. *The Review of Economic Studies*, 57(3):503–511, 1990.

[150] John Quiggin. Regret theory with general choice sets. *Journal of Risk and Uncertainty*, 8(2):153–165, 1994.

[151] Anatol Rapoport and Albert M Chammah. *Prisoner's dilemma: A study in conflict and cooperation*, volume 165. University of Michigan press, 1965.

[152] Thomas Rauber and Gudula Rünger. *Parallel programming: For multicore and cluster systems*. Springer Science & Business Media, 2013.

[153] Raíl Rojas. Konrad zuse's legacy: the architecture of the z1 and z3. *Annals of the History of Computing, IEEE*, 19(2):5–16, 1997.

[154] Catherine Roucairol and Pierre Hansen. Cut cost minimization in graph partitioning. In *Numerical and Applied Mathematics*, pages 585–587, 1989.

[155] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[156] Martino Ruggiero, Alessio Guerri, Davide Bertozzi, Francesco Poletti, and Michela Milano. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pages 3–8. European Design and Automation Association, 2006.

[157] Ponnuswamy Sadayappan, Fikret Ercal, and J. Ramanujam. Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel computing*, 13(1):1–16, 1990.

[158] Ponnuswany Sadayappan and Fikret Ercal. Nearest-neighbor mapping of finite element graphs onto processor meshes. *Computers, IEEE Transactions on*, 100(12):1408–1424, 1987.

[159] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[160] Harvind Samra, Zhi Ding, and Peter M Hahn. Optimal symbol mapping diversity for multiple packet transmissions. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 4, pages IV–181. IEEE, 2003.

[161] Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *LNCS*, pages 164–175. Springer, 2013.

[162] Leonard J. Savage. *The foundations of statistics*. Courier Corporation, 1972.

[163] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

[164] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*, pages 93–100, 1985.

[165] Robert R. Schaller. Moore's law: past, present and future. *Spectrum, IEEE*, 34(6):52–59, 1997.

[166] Niranjan Nath Segal. Signaling system seven distributed call terminating processor, October 21 1997. US Patent 5,680,437.

[167] Lloyd S. Shapley. Cores of convex games. *International journal of game theory*, 1(1):11–26, 1971.

[168] Chien-Chung Shen and Wen-Hsiang Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *Computers, IEEE Transactions on*, 100(3):197–203, 1985.

[169] Itamar Simonson. The influence of anticipating regret and responsibility on purchase decisions. *Journal of Consumer Research*, pages 105–118, 1992.

[170] J.-B. Sinclair. Efficient computation of optimal assignments for distributed tasks. *Journal of Parallel and Distributed Computing*, 4(4):342–362, 1987.

[171] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, page 1. ACM, 2013.

[172] Oliver Sinnen. *Task scheduling for parallel systems*, volume 60. John Wiley & Sons, 2007.

[173] Oliver Sinnen and Leonel Sousa. A classification of graph theoretic models for parallel computing. Technical report, Technical report, Instituto Superior Tecnico, Technical University of Lisbon, Portugal, 1999.

[174] Renaud Sirdey. *Contributions à l'optimisation combinatoire pour l'embarqué: des autocommutateurs cellulaires aux microprocesseurs massivement parallèles*. PhD thesis, Université de Technologie de Compiègne, 2011.

[175] Renaud. Sirdey, Vincent. David, Thiery. Dubrulle, Paul .and Goubier, and Stéphane. Louise. Une procédure de construction des binaires pour un cluster mppa sans translation d'adresse. rapport technique dtsi. Technical report, SARC/09-205, Commissariata l'Énergie Atomique, 2009.

[176] Oana Stan. *Placement of tasks under uncertainty on massively multicore architectures*. PhD thesis, Université de Technologie de Compiègne, 2013.

[177] Oana Stan, Renaud Sirdey, Jacques Carlier, and Dritan Nace. A grasp for placement and routing of dataflow process networks on many-core architectures. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 219–226. IEEE, 2013.

[178] Oana Stan, Renaud Sirdey, Jacques Carlier, and Dritan Nace. A grasp metaheuristic for the robust mapping and routing of dataflow process networks on manycore architectures. *4OR*, pages 1–26, 2015.

[179] Harold S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *Software Engineering, IEEE Transactions on*, (1):85–93, 1977.

[180] Robert Sugden. Regret, recrimination and rationality. *Theory and Decision*, 19(1):77–99, 1985.

[181] Robert Sugden. Regret, recrimination and rationality. *Theory and Decision*, 19(1):77–99, 1985.

[182] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[183] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *Compiler Construction*, pages 179–196. Springer, 2002.

[184] Aleksandar Trifunovic and William J Knottenbelt. Parkway 2.0: A parallel multilevel hypergraph partitioning tool. In *Computer and Information Sciences-ISCIS 2004*, pages 789–800. Springer, 2004.

[185] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[186] Eric Van Dijk and Marcel Zeelenberg. On the psychology of 'if only': Regret and the comparison between factual and counterfactual outcomes. *Organizational Behavior and Human Decision Processes*, 97(2):152–160, 2005.

[187] John Von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[188] John Von Neumann. John von neumann. *American Mathematical Soc*, 1988.

[189] John Von Neumann and Oskar Morgenstern. Theory of games and economic behavior. *Bull. Amer. Math. Soc*, 51(7):498–504, 1945.

[190] C. Walshaw and M. Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. *SIAM J. Sci. Comput.*, 22(1):63–80, 2000.

[191] C. Walshaw and M. Cross. Parallel Optimisation Algorithms for Multilevel Mesh Partitioning. *Parallel Comput.*, 26(12):1635–1660, 2000.

[192] C. Walshaw and M. Cross. Multilevel Mesh Partitioning for Heterogeneous Communication Networks. *Future Generation Comput. Syst.*, 17(5):601–623, 2001.

[193] C. Walshaw and M. Cross. JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview. In F. Magoules, editor, *Mesh Partitioning Techniques and Domain Decomposition Techniques*, pages 27–58. Civil-Comp Ltd., 2007. (Invited chapter).

[194] C. Walshaw, M. Cross, R. Diekmann, and F. Schlimbach. Multilevel Mesh Partitioning for Optimising Domain Shape. *Intl. J. High Performance Comput. Appl.*, 13(4):334–353, 1999.

[195] Chris Walshaw. The graph partitioning archive, 2002.

[196] Benjamin Wells. Advances in i/o, speedup, and universality on colossus, an unconventional computer. In *Unconventional Computation*, pages 247–261. Springer, 2009.

[197] Rui Xu, Donald Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.